به نام دادار دادآفرین

مینی پروژه اول درس مبانی سیستم‌های هوشمند(قسمت اول)

استاد درس: دکتر مهدی علیاری

گردآورنده: امیر جهانگرد تکالو

شماره دانشجویی: ۹۹۲۵۲۹۳

۱۳۰۷

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق

سوال ۱:

۱_۱:

کد مرتبط با این بخش:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
```

در این قسمت ابتدا کتابخانه numpy و matplotlib که قبلا نیز از آنها استفاده می‌کردیم را فراخوانی می‌کنیم. می دانیم که pyplot. برای رسم نمودارها و تصاویر و lines. برای تعریف خطوط در تصاویر استفاده می‌شود. سپس از دیتاست‌های scikit-learn استفاده می‌کنیم که به صورت sklearn.datasets نوشته می‌شود. از ۳ کتابخانه make_classification، make_blobs و make_circles برای ایجاد دیتاست‌های مصنوعی برای آموزش مدل-های یادگیری ماشین استفاده می‌شود

```
# part1
X , y = make_classification(n_samples=1000,
                            n_features=2,
                            n_redundant=0,
                            n_classes=2,
                            n_clusters_per_class=1,
                            class_sep=2,
                            random_state=93)
colors = np.array(['red','blue'])
plt.scatter(X[:,0],X[:,1],c=colors[y])
plt.show()
```

این کد یک دیتاست مصنوعی با استفاده از تابع `make_classification` ایجاد می‌کند .

به بررسی این تابع آماده در پایتون می‌پردازیم:

۱ `n_samples` تعداد نمونه‌های داده است.

۲ `n_features` تعداد ویژگی‌های هر نمونه است.

۳ `n_redundant` تعداد ویژگی‌های اضافی است که برای ایجاد تشابه بین نمونه‌ها اضافه شده است.

۴ `n_classes` تعداد کلاس‌های داده است.

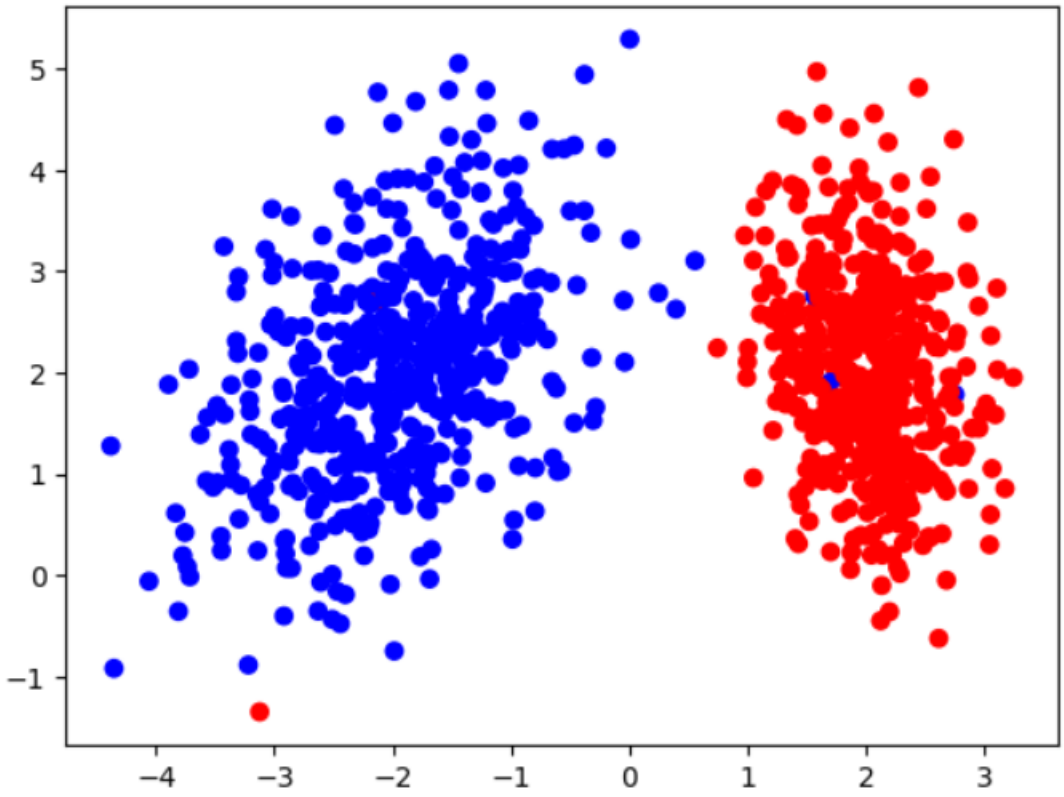
۵ `n_clusters_per_class` تعداد خوشه‌هایی است که برای هر کلاس ایجاد می‌شود.

۶ `class_sep` فاصله بین خوشه‌های هر کلاس است.

۷ `random_state` برای تولید داده‌های تصادفی یک عدد اولیه تعیین می‌کند.

سپس داده‌ها با استفاده از `plt.scatter` رسم شده و با استفاده از `colors[y]` به هر نقطه رنگی نسبت داده می‌شود که `y` برچسب کلاس هر نقطه است. سپس با استفاده از `plt.show()` نمودار رسم شده نمایش داده می‌شود.

نتیجه خروجی بدین صورت است:

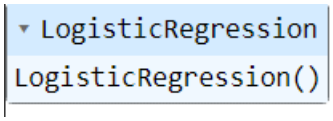


۱_۲ و ۱_۳:

```
# part 2 & 3
model=LogisticRegression()
model.fit(X, y)
```

این قسمت از کد برای ایجاد یک مدل رگرسیون لجستیک با استفاده از داده‌های X و برچسب‌های y استفاده می‌شود. با استفاده از `model.fit(X, y)`، مدل براي یادگیری با داده‌ها آموزش داده می‌شود.

خروجی مشاهده‌شده:




```
X1_min , X2_min = X.min(0)
X1_max , X2_max = X.max(0)
n=1000
x1r = np.linspace(X1_min ,X1_max,n)
x2r = np.linspace(X2_min ,X2_max,n)
X1m, X2m = np.meshgrid(x1r,x2r)
Xm = np.stack((X1m.flatten(),X2m.flatten()),axis=1)
ym = model.decision_function(Xm)
plt.scatter(X[:,0],X[:,1],c=colors[y])
plt.contour(X1m, X2m, ym.reshape(X1m.shape), levels=[0])
plt.show()
```

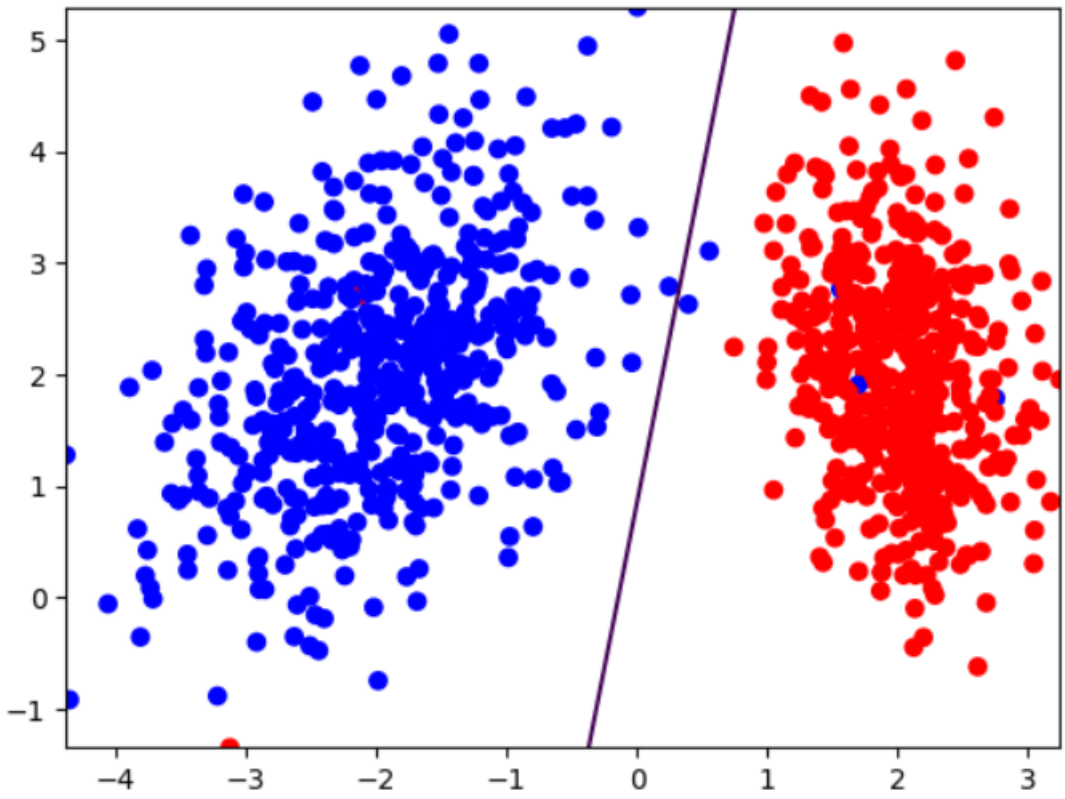
این قسمت از کد برای رسم مرز تصمیم‌گیری مدل رگرسیون لجستیک بر روی داده‌ها استفاده می‌شود.

در اینجا، محدوده‌ای از داده‌ها با استفاده از `np.linspace` و `np.meshgrid` تولید می‌شود و سپس برای هر نقطه از این محدوده، مقدار تابع تصمیم گیری مدل با استفاده از `model.decision_function` محاسبه می‌شود.

سپس با استفاده از `plt.contour` خطوط همواره برای مقادیر مختلف تابع تصمیم گیری رسم می‌شود. این خطوط مرز تصمیم گیری برای دسته‌بندی داده‌ها توسط مدل را نشان می‌دهند.

در نهایت با استفاده از `plt.show()` نمودار نهایی رسم شده و نمایش داده می‌شود.

خروجی این قسمت از کد را مشاهده می‌کنیم:

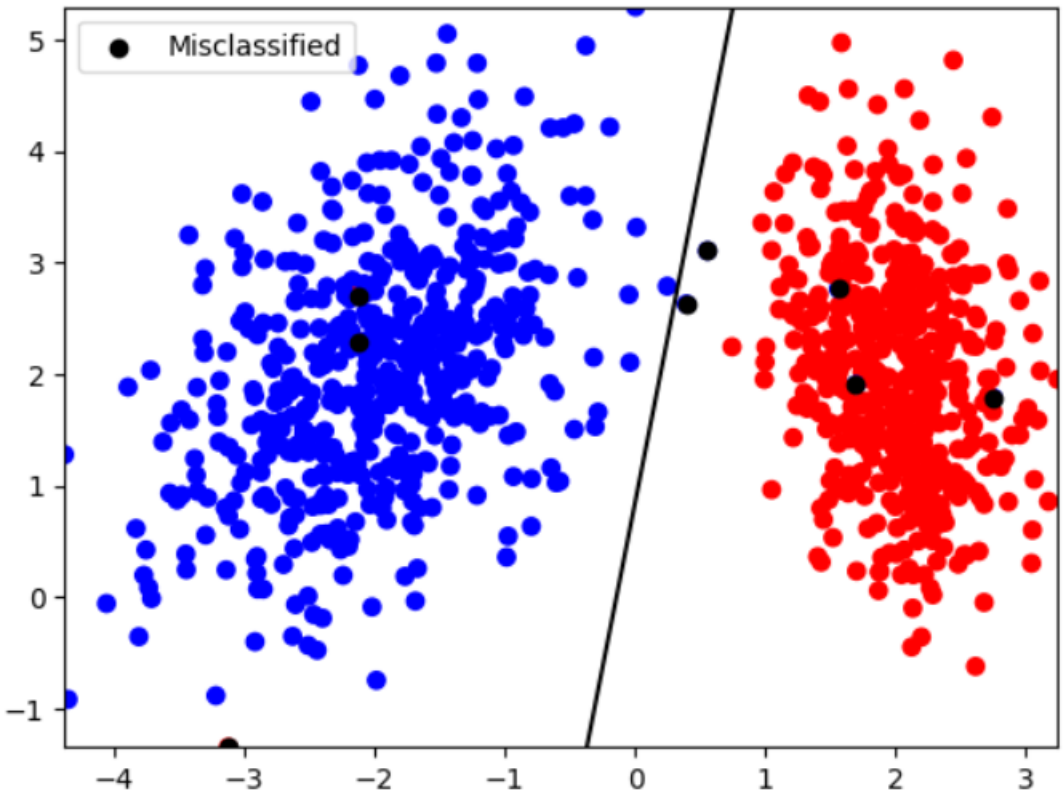



```
colors = np.array(['red', 'blue'])

plt.scatter(X[:, 0], X[:, 1], c=colors[y])

y_pred = model.predict(X)

wrong_indices = np.where(y != y_pred)[0]
plt.scatter(X[wrong_indices, 0], X[wrong_indices, 1], c='black',
label='Misclassified')
plt.contour(X1m, X2m, ym.reshape(X1m.shape), levels=[0], colors='black')

plt.legend()
plt.show()
```

این کد یک نمودار scatter plot ایجاد می‌کند. ابتدا دو رنگ red و blue را به صورت یک آرایه numpy تعریف کرده و سپس از آن برای نمایش دادن نقاط استفاده می‌کند .

سپس مدل را بر روی داده‌ها predict می‌کند و نقاطی که اشتباه predict شده‌اند را با رنگ سیاه نمایش می‌دهد. در نهایت یک contour plot از مرز تصمیم را روی داده‌ها نیز رسم می‌کند.

نمایش خروجی:



در ادامه برای نمایش بهتر می‌توان از دستور زیر نیز استفاده کرد:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

این بخش از کد برای ایجاد یک مدل شبکه عصبی با استفاده از کتابخانه Keras است. ابتدا از Keras وابستگی‌های لازم را وارد می‌کنیم. سپس یک مدل Sequential ایجاد می‌کنیم که به ترتیب لایه‌های مختلف را به صورت پشت سرهم قرار می‌دهد. در اینجا از Dense برای ایجاد لایه‌های کاملاً متصل استفاده می‌شود. همچنین از Adam به عنوان یک الگوریتم بهینه‌سازی برای آموزش مدل استفاده می‌شود.

```
model = Sequential()
model.add(Dense(units = 1 , input_shape = (2,) , activation = 'sigmoid'))
adam = Adam(learning_rate = 0.3)
```

```
model.compile(adam , loss = 'binary_crossentropy' , metrics =
['accuracy'])
h = model.fit(x = X , y = y , verbose = 1 , batch_size = 50 , epochs =
1000 , shuffle = True)
plt.plot(h.history['accuracy'])
plt.title('accuracy')
plt.xlabel('epoch')
plt.legend(['accuracy'])
```

در این قسمت ابتدا یک شیء از کلاس Sequential ایجاد می‌شود. سپس با استفاده از model.add یک لایه Dense با یک واحد و فعال‌ساز sigmoid به مدل اضافه می‌شود.

سپس با استفاده از Adam یک شیء از کلاس Adam به عنوان الگوریتم بهینه‌سازی ایجاد می‌شود. سپس با استفاده از model.compile، مدل آماده آموزش می‌شود.

در اینجا، به عنوان تابع هزینه binary_crossentropy و به عنوان معیار عملکرد accuracy انتخاب شده است.

سپس با استفاده از model.fit، مدل با داده‌ها آموزش داده می‌شود.

در نهایت با استفاده از plt.plot، تاریخچه دقت مدل در هر دوره آموزش رسم می‌شود.

در نتیجه‌ای که مشاهده می‌کنیم درصد خطا و صحت را نیز می‌بینیم که با افزایش هر Epoch مقدار accuracy بالاتر می‌رود و مقدار loss پایین می‌آید.

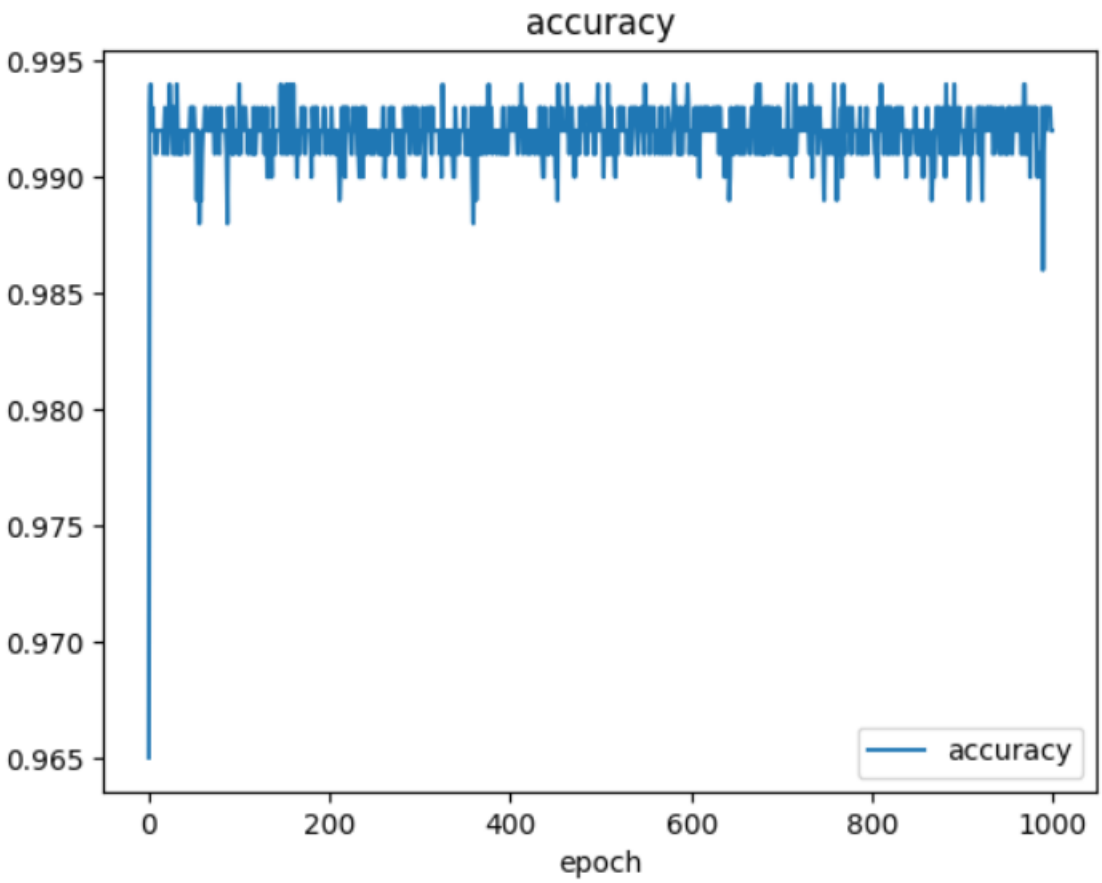
با توجه به اینکه ۱۰۰۰ Epoch داده‌ایم پس مقدار کمی از آنرا بررسی می‌کنیم:

.

```
Epoch 1/1000
20/20 [==============================] - 1s 4ms/step - loss: 0.1126 - accuracy: 0.9650
Epoch 2/1000
20/20 [==============================] - 0s 6ms/step - loss: 0.0613 - accuracy: 0.9930
Epoch 3/1000
20/20 [==============================] - 0s 6ms/step - loss: 0.0574 - accuracy: 0.9940
Epoch 4/1000
20/20 [==============================] - 0s 5ms/step - loss: 0.0561 - accuracy: 0.9920
Epoch 5/1000
20/20 [==============================] - 0s 3ms/step - loss: 0.0562 - accuracy: 0.9930
Epoch 6/1000
20/20 [==============================] - 0s 5ms/step - loss: 0.0559 - accuracy: 0.9920
Epoch 7/1000
20/20 [==============================] - 0s 7ms/step - loss: 0.0535 - accuracy: 0.9920
Epoch 8/1000
20/20 [==============================] - 0s 4ms/step - loss: 0.0559 - accuracy: 0.9920
Epoch 9/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0539 - accuracy: 0.9910
Epoch 10/1000
20/20 [==============================] - 0s 4ms/step - loss: 0.0558 - accuracy: 0.9920
```

```
Epoch 992/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0592 - accuracy: 0.9930
Epoch 993/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0552 - accuracy: 0.9920
Epoch 994/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0567 - accuracy: 0.9930
Epoch 995/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0545 - accuracy: 0.9930
Epoch 996/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0553 - accuracy: 0.9930
Epoch 997/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0545 - accuracy: 0.9930
Epoch 998/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0543 - accuracy: 0.9930
Epoch 999/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0567 - accuracy: 0.9920
Epoch 1000/1000
20/20 [==============================] - 0s 2ms/step - loss: 0.0565 - accuracy: 0.9920
```

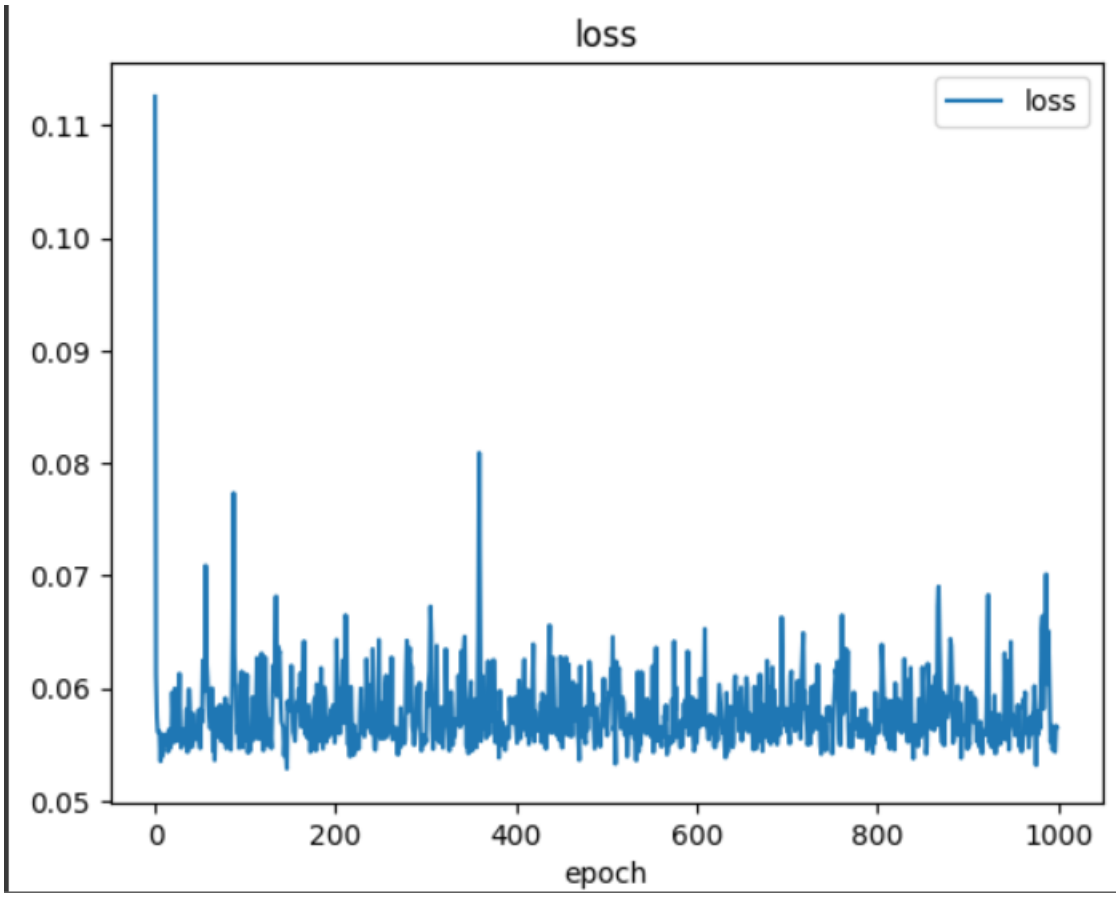
.



نمودار loss را نیز بررسی می‌کنیم:

```
plt.plot(h.history['loss'])
plt.title('loss')
plt.xlabel('epoch')
plt.legend(['loss'])
```

در این برچسب‌گذاری از محور x به عنوان "دوره" و از محور y به عنوان "هزینه" استفاده شده است. این کد به صورت تصویری نمایش می‌دهد که هزینه مدل در هر دوره آموزش چگونه تغییر کرده است. این کار به ما کمک می‌کند تا بفهمیم که آیا مدل در حال آموزش بهتر می‌شود یا نه.



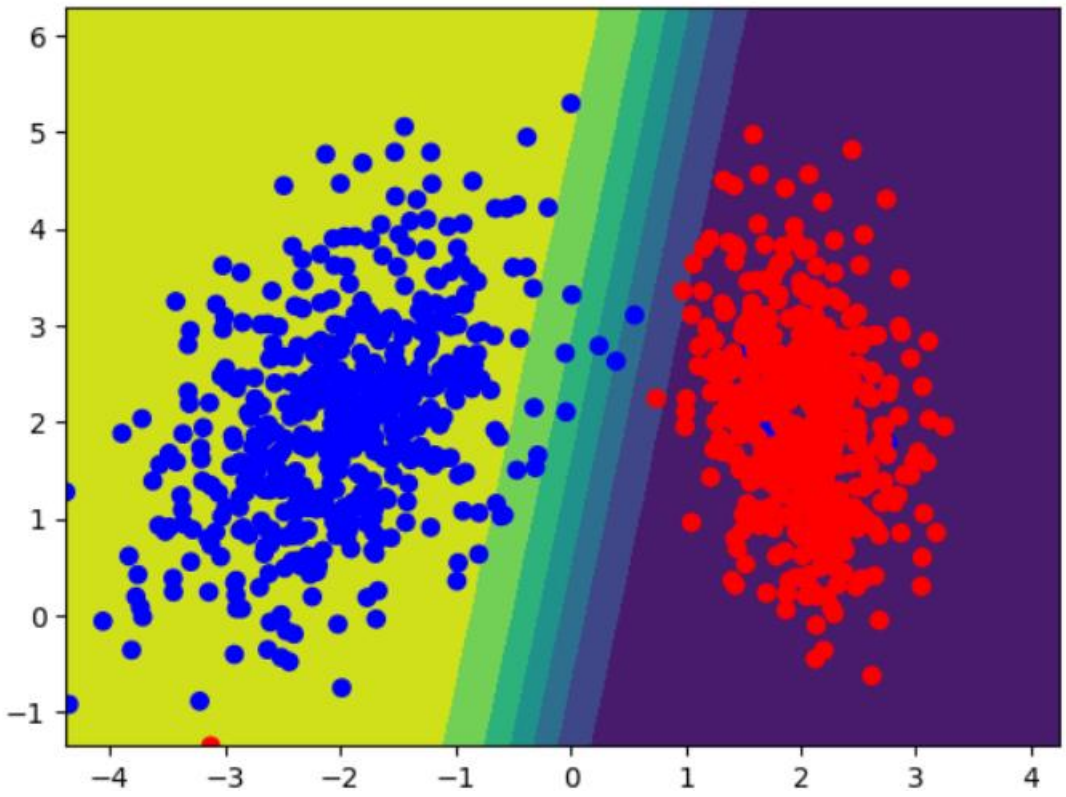

```
def plot_decision_boundry(X,y,model):
    x_span = np.linspace (min(X[:,0]),max(X[:,0])+1 , 50)
    y_span = np.linspace (min(X[:,1]),max(X[:,1])+1 , 50)
    xx , yy = np.meshgrid(x_span , y_span)
    xx_,yy_ = xx.ravel() , yy.ravel()
    grid = np.c_[xx_,yy_]
    pred_func = model.predict(grid)
    z = pred_func.reshape(xx.shape)
    plt.contourf(xx,yy,z)
```

این تابع برای ترسیم مرز تصمیم گیری مدل بر روی داده‌ها استفاده می‌شود. ابتدا یک شبکه از نقاط با استفاده از np.linspace` ایجاد می‌شود. سپس با استفاده از np.meshgrid`، یک شبکه از نقاط ایجاد می‌شود. سپس با استفاده از model.predict`، پیش‌بینی مدل بر روی این نقاط صورت می‌گیرد. در نهایت با استفاده از plt.contourf`، مرز تصمیم‌گیری مدل بر روی نقاط ایجاد شده ترسیم می‌شود.

```
n_samples=1000
plot_decision_boundry(X,y,model)
plt.scatter(X[:n_samples,0],X[:n_samples,1],c=colors[y])
plt.scatter(X[n_samples:,0],X[n_samples:,1],)
```

این کد به شما کمک می‌کند تا مرز تصمیم‌گیری مدل خود را روی داده‌ها نشان دهید. ابتدا با استفاده از "plt.scatter" داده‌ها را روی نمودار نشان می‌دهید. سپس با استفاده از تابع "plot_decision_boundry"، مرز تصمیم‌گیری مدل را روی داده‌ها نشان می‌دهید. این کار به شما کمک می‌کند تا ببینید که چگونه مدل شما داده‌ها را جدا می‌کند و مرز تصمیم‌گیری آن چگونه است.

خروجی بدین صورت است:



۱_۴:

برای سخت‌تر کردن دسته‌بندی دو کلاسه، می‌توان از روش‌های زیر استفاده کرد:

۱ .افزایش تعداد نمونه‌ها: با افزایش تعداد نمونه‌ها در هر کلاس، دسته‌بندی سخت‌تر می‌شود. می‌توانید از تکنیک‌های oversampling مانند SMOTE استفاده کنید تا تعداد نمونه‌های کمترین کلاس را افزایش دهید.

۲ .کاهش تفکیک بین دو کلاس: با کاهش فاصله بین دو کلاس در فضای ویژگی، دسته‌بندی سخت‌تر می‌شود. می‌توانید از روش‌های undersampling مانند RandomUnderSampler استفاده کنید تا تعداد نمونه‌های بیشترین کلاس را کاهش دهید.

۳ . اضافه کردن ویژگی‌های جدید: با اضافه کردن ویژگی‌های جدید به داده‌ها، می‌توانید تفکیک بین دو کلاس را افزایش دهید. می‌توانید از تکنیک‌های feature engineering مانند polynomial features استفاده کنید تا ویژگی‌های ترکیبی ایجاد کنید.

۴ . استفاده از مدل‌های پیچیده‌تر: با استفاده از مدل‌های پیچیده‌تر مانند SVM یاNeural Networks ، می‌توانید دسته‌بندی سخت‌تری را انجام دهید. این مدل‌ها قادرند تفکیک‌های پیچیده‌تر را مدل کنند و در نتیجه دسته‌بندی دقیق‌تری را انجام دهند.

در کد زیر، از روش‌های ۱ و ۴ برای سخت‌تر کردن دسته‌بندی استفاده شده است. با افزایش تعداد نمونه‌ها و استفاده از مدلSVM ، دسته‌بندی چالش برانگیزتری انجام می‌شود:

```
# part 4
from sklearn.datasets import make_classification
from sklearn.svm import SVC
X, y = make_classification(n_samples=1000,
                           n_features=2,
                           n_informative=2,
                           n_redundant=0,
                           n_clusters_per_class=1,
                           class_sep=1.5,
                           random_state=93)

model = SVC(kernel='linear')
model.fit(X, y)

x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x1, x2 = np.meshgrid(np.arange(x1_min, x1_max, 0.02),
                     np.arange(x2_min, x2_max, 0.02))
Z = model.predict(np.c_[x1.ravel(), x2.ravel()])
Z = Z.reshape(x1.shape)

plt.contourf(x1, x2, Z, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
plt.title('SVM Classification')
plt.show()
```

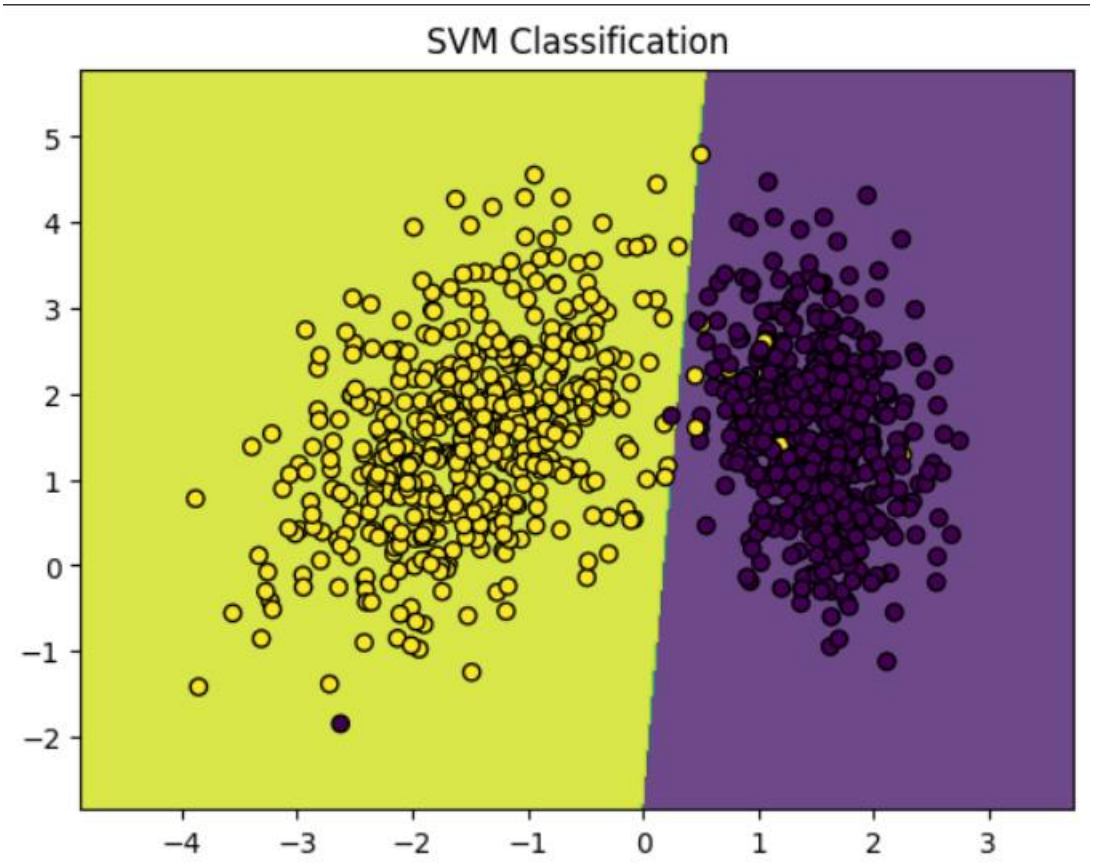
این قسمت برای سخت‌تر شدن دیتاست تولید شده در قسمت ۱ است. این کد یک مدل SVM را با استفاده از داده‌های تصادفی ایجاد می‌کند و سپس یک نمودار contour plot از مرز تصمیم را روی داده‌ها رسم می‌کند .

ابتدا داده‌های تصادفی با استفاده از تابع make_classification ایجاد می‌شوند. سپس مدل SVM با kernel خطی ایجاد می‌شود و بر روی داده‌ها fit می‌شود .

سپس با استفاده ازnp.meshgrid ، فضای دو بعدی از داده‌ها ایجاد شده و با predict کردن مدل روی این فضا، مرز تصمیم به دست می‌آید .

در نهایت با استفاده از contourf و scatter، نمودار نهایی رسم می‌شود.

نمایش خروجی:



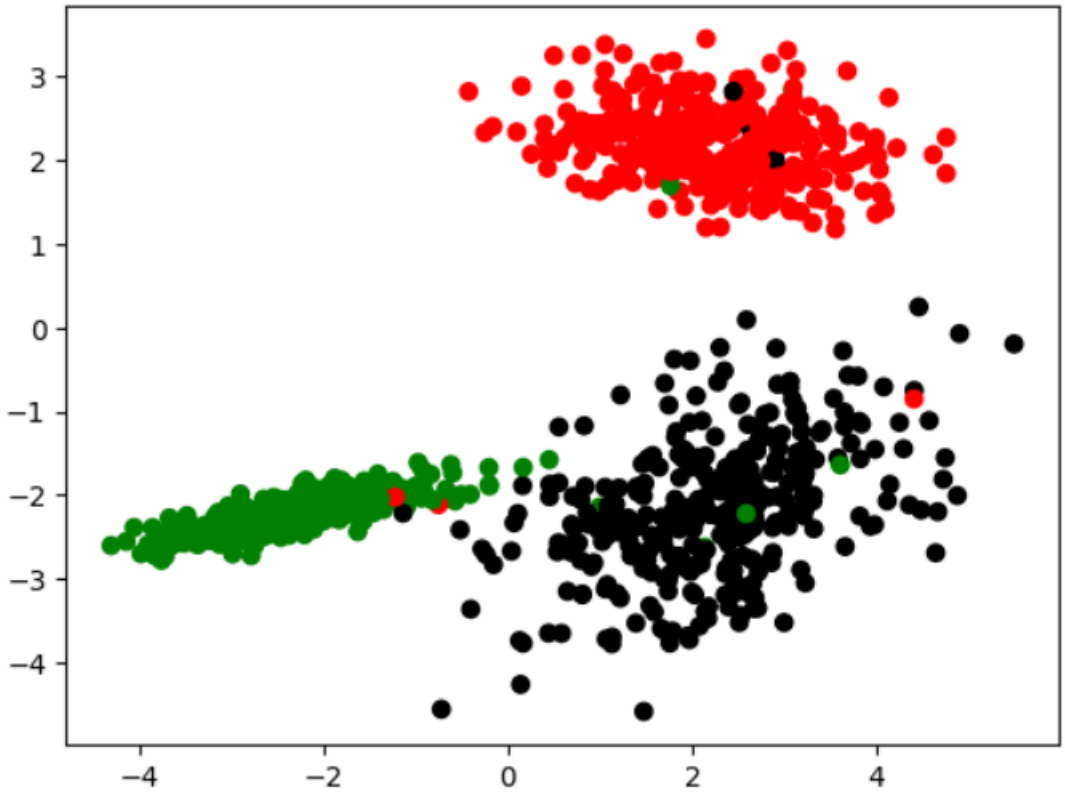
۱_۵:

```
# part 5
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
```

```
from sklearn.datasets import make_classification, make_blobs ,
make_circles
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression , SGDClassifier

X , y = make_classification(n_samples=1000,
                            n_features=2,
                            n_redundant=0,
                            n_classes=3,
                            n_clusters_per_class=1,
                            class_sep=2.2,
                            random_state=93)
colors = np.array(['red','black','green'])
plt.scatter(X[:,0],X[:,1],c=colors[y])
plt.show()
```

همانطور که می‌بینیم در این قسمت از کد ما برای بررسی دسته‌بندی ۳ کلاسه فقط تعداد کلاس‌ها و رنگ‌های ما ۳ تا شدند.

نمایش خروجی:




```
model=LogisticRegression()
model.fit(X, y)
```

```
X1_min, X2_min = X.min(0)
X1_max, X2_max = X.max(0)

n = 500
x1r = np.linspace(X1_min, X1_max, n)
x2r = np.linspace(X2_min, X2_max, n)
X1m, X2m = np.meshgrid(x1r, x2r)
Xm = np.stack((X1m.flatten(), X2m.flatten()), axis=1)
ym = model.predict(Xm)

colors = np.array(['red','black','green'])
plt.scatter(X[:, 0], X[:, 1], c=colors[y])
plt.contour(X1m, X2m, ym.reshape(X1m.shape), levels=[0, 1, 2],
colors='blue')
plt.show()
```

این قسمت از کد، مرز تصمیم را روی داده‌های تصادفی که در قسمت قبل ایجاد شده بود، رسم می‌کند.

ابتدا با استفاده از min وmax ، حداقل و حداکثر مقادیر برای هر ویژگی از داده‌ها به دست می‌آیند.

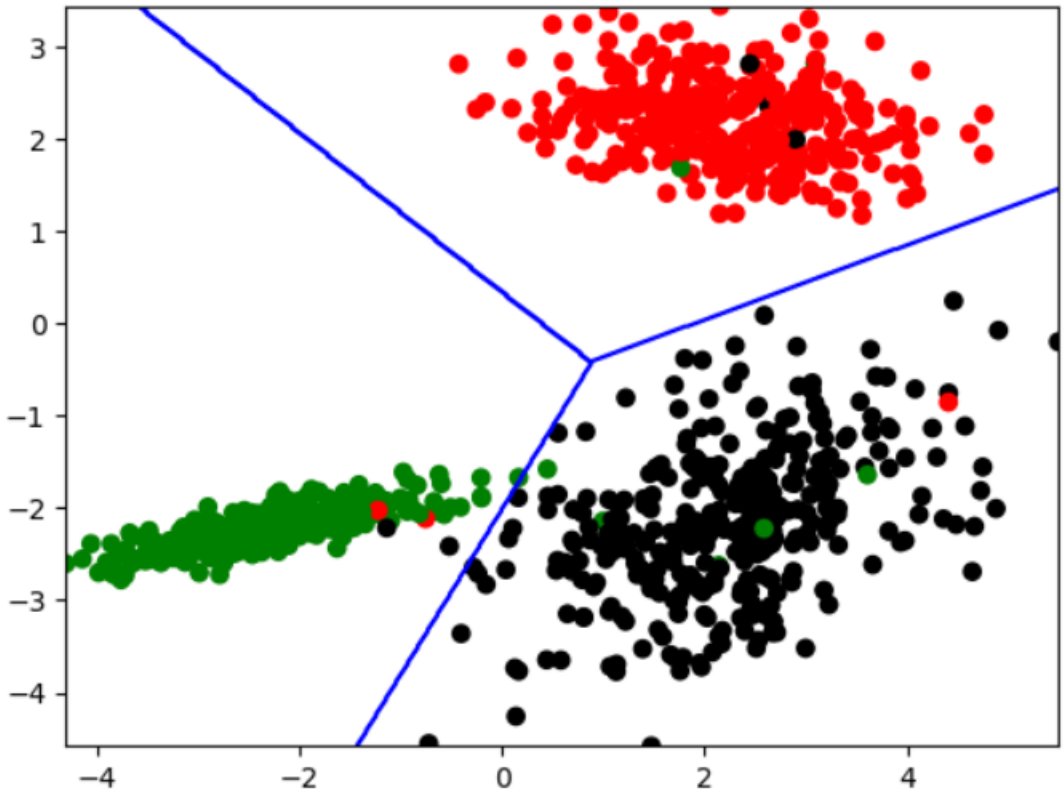
سپس با استفاده ازlinspace ، nنقطه برای هر ویژگی از داده‌ها ایجاد می‌شود و با استفاده ازmeshgrid ، فضای دو بعدی از داده‌ها ایجاد می‌شود.

سپس با استفاده از predict کردن مدل SVM روی این فضا، مرز تصمیم به دست می‌آید.

در نهایت با استفاده از plt.scatter و plt.contour، داده‌ها و مرز تصمیم رسم می‌شوند.

تفاوت این قسمت از کد با کد مربوطه به دسته‌بندی ۲ کلاسه در این است که در بخش کانتور باید ۳ تا سطح برای تفکیک‌سازی در نظر بگیریم، زیرا قرار است ۳ تا خط رسم کنیم.

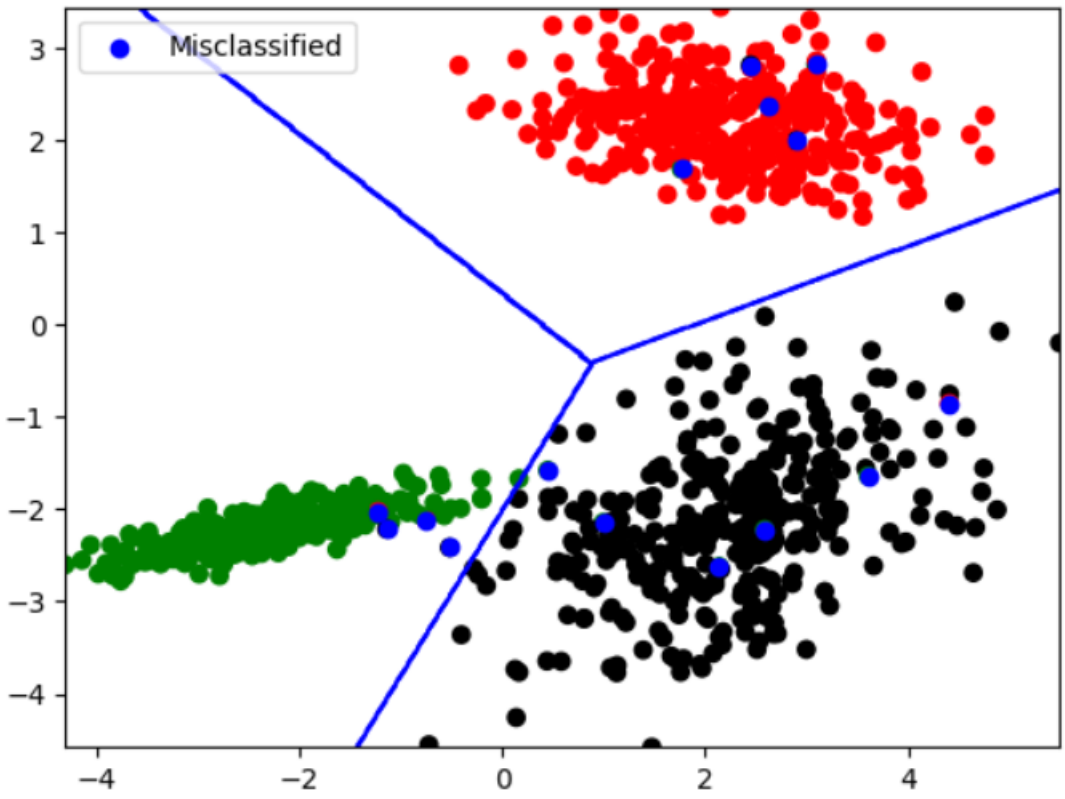
نمایش خروجی:




```
y_pred = model.predict(X)
colors = np.array(['red','black','green'])
plt.scatter(X[:, 0], X[:, 1], c=colors[y])
plt.contour(X1m, X2m, ym.reshape(X1m.shape), levels=[0, 1, 2],
colors='blue', linestyles='dashed')

wrong_indices = np.where(y != y_pred)[0]
plt.scatter(X[wrong_indices, 0], X[wrong_indices, 1], c='blue',
label='Misclassified')

plt.contour(X1m, X2m, ym.reshape(X1m.shape), levels=[0, 1, 2],
colors='blue')
plt.legend()
plt.show()
```

در نهایت همانند دسته‌بندی ۲ کلاسه، داده‌های پرت که اشتباه طبقه‌بندی شده‌اند را تفکیک می‌کنیم و با رنگ آبی نمایش می‌دهیم.

نمایش خروجی:



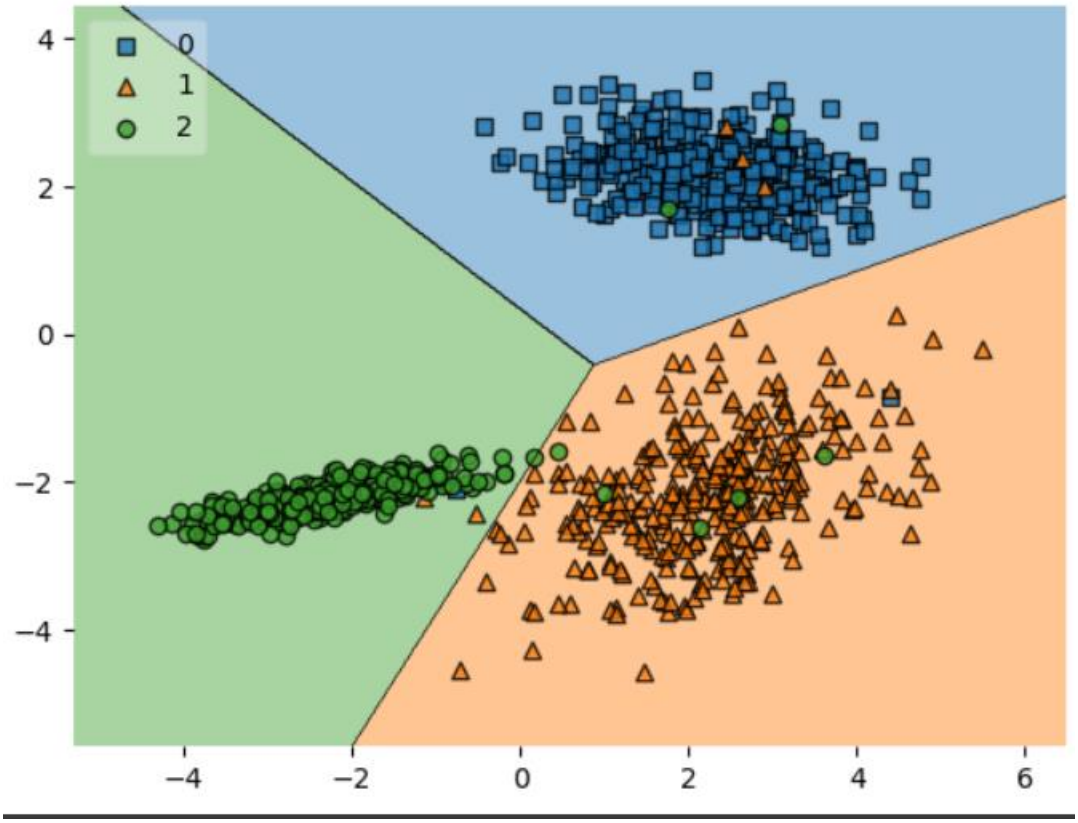
حال برای حل این قسمت از دستورات آماده پایتونی استفاده می‌کنیم:

```
from mlxtend.plotting import plot_decision_regions
```

کتابخانه mlxtend دارای یک تابع به نام plot_decision_regions است که برای ترسیم مرز تصمیم برای مدل‌های دسته‌بندی استفاده می‌شود. این تابع می‌تواند برای ترسیم مرز تصمیم برای مدل‌های دسته‌بندی چند کلاسه و دو کلاسه استفاده شود و به آسانی می‌تواند در تحلیل و یادگیری داده‌ها مورد استفاده قرار گیرد.

```
plot_decision_regions(X, y, clf=model,legend=2)
```

نمایش خروجی:



برای دسته‌بندی ۳ کلاسه نیز می‌توان از کتابخانه‌های مربوط به شبکه‌های عصبی استفاده کرد. با این تفاوت که یک کتابخانه جدید زیر اضافه می‌شود:

```
from keras.utils import to_categorical
```

کد بالا از کتابخانه Keras، تابع to_categorical را وارد می‌کند. این تابع برای تبدیل برچسب‌ها به فرمت یک‌به‌یک (one-hot)استفاده می‌شود. در فرمت یک‌به‌یک، هر برچسب به یک بردار دودویی تبدیل می‌شود که در آن تمام عناصر برابر با صفر هستند، به‌جز یک عنصر که متناظر با برچسب مورد نظر یک است. این فرمت برای آموزش مدل‌های عمیق و شبکه‌های عصبی بسیار مفید است.

```
y_cat = to_categorical(y, 3)
model = Sequential()
model.add(Dense(3, input_shape=(2,), activation='softmax'))
model.compile(Adam(learning_rate=0.1), 'categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(X, y_cat, verbose=1, batch_size = 50, epochs=100)
```

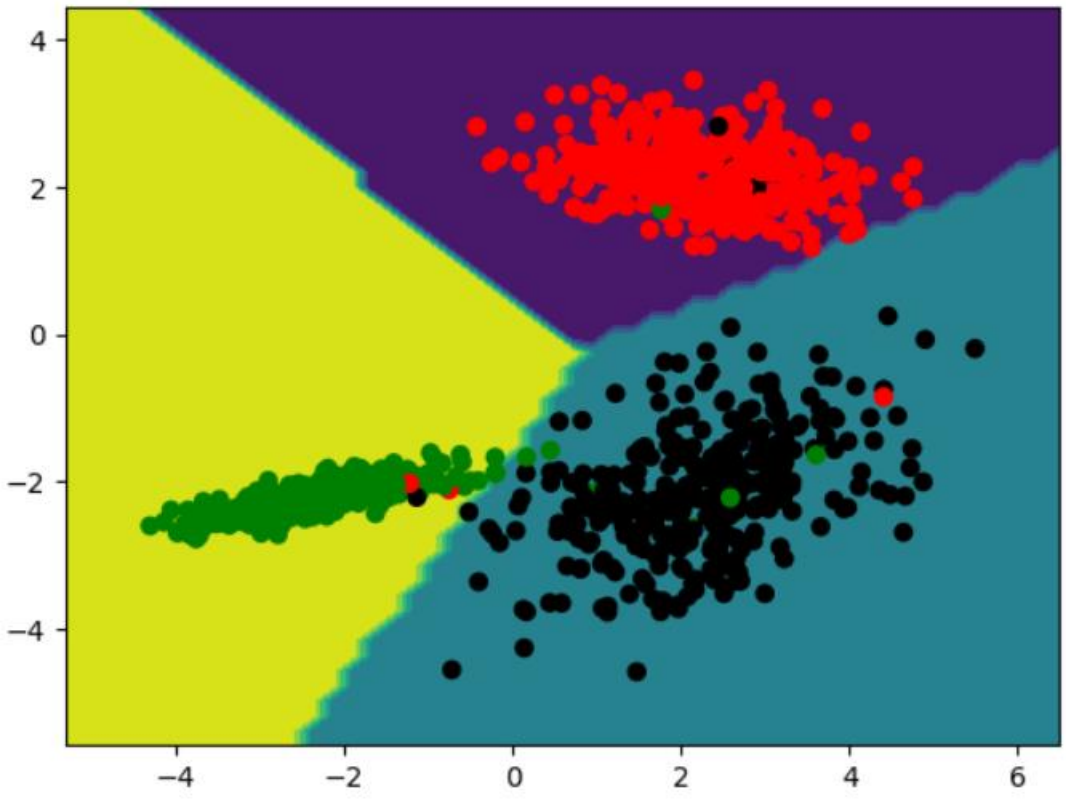
این کد از کتابخانه Keras استفاده می‌کند تا یک مدل شبکه عصبی را برای دسته‌بندی داده‌ها ایجاد کند. ابتدا داده‌های برچسب‌دار ورودی (X) و برچسب‌های آن‌ها (y) را به تابع to_categorical ارسال می‌کند تا برچسب‌ها به فرمت یک‌به‌کی تبدیل شوند. سپس یک مدل شبکه عصبی با یک لایه ورودی دارای ۲ نورون، یک لایه خروجی دارای ۳ نورون و تابع فعال‌ساز softmax ایجاد می‌کند. سپس از الگوریتم بهینه‌ساز Adam با نرخ یادگیری ۰/۱ و تابع هزینه categorical_crossentropy برای آموزش مدل استفاده می‌کند. در نهایت با استفاده از داده‌ها و برچسب‌های یک‌به‌کی آموزش مدل را با ۱۰۰ دور آموزش می‌دهد.

```
def plot_multiclass_decision_boundary(X, y, model):
    x_span = np.linspace(min(X[:,0]) - 1, max(X[:,0]) + 1)
    y_span = np.linspace(min(X[:,1]) - 1, max(X[:,1]) + 1)
    xx, yy = np.meshgrid(x_span, y_span)
    grid = np.c_[xx.ravel(), yy.ravel()]
    pred_func = model.predict(grid)
    z = np.argmax(pred_func, axis=1)
    z = z.reshape(xx.shape)
    #z = pred_func.reshape(xx.shape)
    plt.contourf(xx, yy, z)
```

این تابع plot_multiclass_decision_boundary برای ترسیم مرز تصمیم برای یک مدل دسته‌بندی چند کلاسه استفاده می‌شود. ابتدا یک grid از نقاط در فضای ورودی ایجاد می‌شود. سپس مدل به این گریب اعمال شده و پیش‌بینی‌های مختلف برای هر نقطه در گریب به دست می‌آید. سپس از این پیش‌بینی‌ها برای رسم مرز تصمیم با استفاده از تابع contourf استفاده می‌شود. این تابع با استفاده از پیش‌بینی‌های مدل، مرز تصمیم را بر روی نمودار ایجاد می‌کند.

```
plot_multiclass_decision_boundary(X, y_cat, model)
plt.scatter(X[:1000,0],X[:1000,1],c=colors[y])
plt.scatter(X[1000:,0],X[1000:,1],)
```

نمایش خروجی:


4
2
0
−2
−4
−4
−2
0
2
4
6

سوال ۲:

۲_۱:

داده ها از تصاویری که از نمونه های واقعی و جعلی شبیه اسکناس گرفته شده بودند استخراج شد. برای دیجیتالی کردن، از یک دوربین صنعتی که معمولاً برای بازرسی چاپ استفاده می شود استفاده می شود. تصاویر نهایی دارای ۴۰۰ در ۴۰۰ پیکسل هستند. با توجه به لنز شی و فاصله تا جسم مورد بررسی، تصاویری در مقیاس خاکستری با وضوح حدود ۶۶۰ نقطه در اینچ به دست آمد. ابزار تبدیل موجک برای استخراج ویژگی ها از تصاویر استفاده شد.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Computer Science	Classification
Feature Type	**# Instances**	**# Features**
Real	1372	-

```
# part 1
#https://drive.google.com/file/d/172qsBrIM5UpTaRXWo2Yg6NN1fgw10t5w/view?usp=sharing
!gdown 172qsBrIM5UpTaRXWo2Yg6NN1fgw10t5w
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression , SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

```
read_file = pd.read_csv (r'/content/data_banknote_authentication.txt')
read_file.to_csv (r'/content/data_banknote_authentication.csv')
df = pd.read_csv("/content/data_banknote_authentication.csv")
df
```

ابتدا فایل متنی 'data_banknote_authentication.txt' را با استفاده از کتابخانه pandas می‌خوانیم و سپس آن را به فرمت CSV ذخیره می‌کنیم. سپس داده‌ها را از فایل CSV می‌خوانیم و آنها را در متغیر df ذخیره می‌کنیم.

حال خروجی ما در پایتون نمایان شد:

	Unnamed: 0	3.6216	8.6661	-2.8073	-0.44699	0
0	0	4.54590	8.16740	-2.4586	-1.46210	0
1	1	3.86600	-2.63830	1.9242	0.10645	0
2	2	3.45660	9.52280	-4.0112	-3.59440	0
3	3	0.32924	-4.45520	4.5718	-0.98880	0
4	4	4.36840	9.67180	-3.9606	-3.16250	0
...	...	...	...	...	...	...
1366	1366	0.40614	1.34920	-1.4501	-0.55949	1
1367	1367	-1.38870	-4.87730	6.4774	0.34179	1
1368	1368	-3.75030	-13.45860	17.5932	-2.77710	1
1369	1369	-3.56370	-8.38270	12.3930	-1.28230	1
1370	1370	-2.54190	-0.65804	2.6842	1.19520	1

1371 rows × 6 columns

۲_۲:

اهمیت شافل کردن دیتا ها در یادگیری ماشین به شرح زیر است:

- جلوگیری از بروز الگوهای کاذب: شافل کردن دیتا ها باعث می شود که الگوریتم یادگیری ماشین الگوهای کاذبی را در دیتا ها تشخیص ندهد. این الگوهای کاذب می توانند ناشی از ترتیب خاصی از داده ها در مجموعه داده باشند.

- بهبود عملکرد الگوریتم: شافل کردن دیتا ها می تواند به بهبود عملکرد الگوریتم یادگیری ماشین کمک کند. این به این دلیل است که شافل کردن باعث می شود که الگوریتم به طور مساوی از تمام داده ها استفاده کند و از تأثیر داده های نادرست یا غیرعادی جلوگیری کند.

- افزایش سرعت یادگیری: شافل کردن دیتا ها می تواند به افزایش سرعت یادگیری الگوریتم یادگیری ماشین کمک کند. این به این دلیل است که شافل کردن باعث می شود که الگوریتم به طور مساوی از تمام داده ها استفاده کند و از تکرار داده ها جلوگیری کند.

```
# part 2
file = pd.read_csv (r'/content/data_banknote_authentication.txt')
#file = read_file.to_csv (r'/content/data_banknote_authentication.csv')
headerlist = ['part1' , 'part2','part3','part4','part5']
file.to_csv("/content/data_banknote_authentication.csv" ,header =
headerlist)
df = pd.read_csv("/content/data_banknote_authentication.csv")
df = shuffle(df)
df
```

ابتدا فایل متنی 'data_banknote_authentication.txt' را با استفاده از کتابخانه pandas می‌خوانیم و سپس آن را به فرمت CSV ذخیره می‌کنیم. سپس داده‌ها را از فایل CSV می‌خوانیم و آنها را با استفاده از تابع shuffle از کتابخانهpandas ، تصادفی می‌کنیم. در ضمن در headerlist ما عناوین ستون‌ها را تعریف کردیم.

حال خروجی را مشاهده می‌کنیم:

	Unnamed: 0	part1	part2	part3	part4	part5
737	737	0.92703	9.43180	-0.66263	-1.67280	0
1359	1359	-0.24745	1.93680	-2.46970	-0.80518	1
1280	1280	-2.79080	-5.71330	5.95300	0.45946	1
963	963	-1.41060	-7.10800	5.64540	0.31335	1
1063	1063	-3.69610	-13.67790	17.57950	-2.61810	1
...	...	...	...	...	...	...
711	711	4.79650	6.98590	-1.99670	-0.35001	0
1362	1362	-1.16670	-1.42370	2.92410	0.66119	1
1078	1078	0.12126	0.22347	-0.47327	0.97024	1
1211	1211	-2.45600	-0.24418	1.40410	-0.45863	1
985	985	0.84546	3.48260	-3.63070	-1.39610	1

1371 rows × 6 columns

```
X = df[['part1' , 'part2','part3','part4']].values
y = df[['part5']].values
X ,y
```

در این بخش از کد، داده‌های موجود در dataframe به نام df را برای استفاده در یک مدل یادگیری ماشینی آماده می‌کنیم. ابتدا داده‌های ستون‌های 'part1'، 'part2'، 'part3' و 'part4' را به عنوان ورودی (X) و داده‌های ستون 'part5' را به عنوان خروجی (y) انتخاب می‌کنیم. سپس مقادیر X و y را چاپ می‌کنیم تا اطمینان حاصل کنیم که داده‌ها به درستی بارگیری شده‌اند.

وضعیت خروجی را مشاهده می‌کنیم:

```
(array([[ 0.44125,  2.9487 ,  4.3225 ,  0.7155 ],
        [-1.8554 , -9.6035 ,  7.7764 , -0.97716],
        [-1.1497 ,  1.2954 ,  7.701  ,  0.62627],
        ...,
        [ 4.6352 , -3.0087 ,  2.6773 ,  1.212  ],
        [ 3.1887 , -3.4143 ,  2.7742 , -0.2026 ],
        [ 0.26637,  0.73252, -0.67891,  0.03533]]),
 array([[0],
        [1],
        [0],
        ...,
        [0],
        [0],
        [1]]))
```

```
X.shape , y.shape
```

```
((1371, 4), (1371, 1))
```

```
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size =
0.2)
x_train.shape , x_test.shape , y_train.shape , y_test.shape
```

در این بخش از کد، داده‌های ورودی X و خروجی y را به دو بخش آموزش و آزمون با نسبت ۸۰٪ به ۲۰٪ تقسیم می‌کنیم. سپس اندازه‌ی داده‌های آموزش و آزمون را چاپ می‌کنیم تا اطمینان حاصل کنیم که این تقسیم به درستی انجام شده است:

```
((1096, 4), (275, 4), (1096, 1), (275, 1))
```

۳_۲:

```
def sigmoid(score):
    return 1/(1+np.exp(-score))
```

در این بخش از کد، یک تابع سیگموید تعریف شده است که ورودی یک امتیاز یا نمره را می‌گیرد و مقدار سیگموید این عدد را محاسبه می‌کند و برمی‌گرداند. تابع سیگموید معمولا در مدل‌های یادگیری ماشین برای تبدیل امتیازها یا امتیازات به احتمالات استفاده می‌شود. این تابع از کتابخانه NumPy برای محاسبه توان و تابع exponential استفاده می‌کند.

```
def logistic_regression(x , w):
  p = sigmoid(x @ w)
  return p
```

در این بخش از کد، یک تابع رگرسیون لجستیک پیاده‌سازی شده است. ورودی‌های این تابع شامل ماتریس x و بردار وزن w می‌باشد. ابتدا ماتریس x با بردار وزن w ضرب داخلی می‌شود و سپس نتیجه‌ی این ضرب داخلی به تابع سیگموید sigmoid داده می‌شود تا احتمال متناظر با ورودی‌های x و w محاسبه شود.

```
def calculate_error(y,p):
    cross_entropy = -(np.mean(y*np.log(p) + (1-y)*np.log(1-p)))
    return cross_entropy
```

در این بخش از کد، یک تابع برای محاسبه‌ی خطای مدل با استفاده از تابع هزینه cross-entropy پیاده‌سازی شده است. ورودی‌های این تابع شامل بردار y که برچسب‌های واقعی داده‌ها را نشان می‌دهد و بردار p که نتیجه تابع سیگموید توسط مدل را نشان می‌دهد، می‌باشد. ابتدا بردار y و بردار p به صورت ماتریسی در می‌آیند و سپس با استفاده از فرمول cross-entropy، خطای مدل محاسبه می‌شود.

```
def gradient(x , y  ,p):
  grads = (x.T @(p - y)) / len(y)
  return grads
```

تابع gradient در پایتون، برای محاسبه ماتریس گرادیان (gradient matrix) به منظور استفاده در الگوریتم‌های یادگیری ماشین و بهبود عملکرد آن‌ها، استفاده می‌شود.

ورودی‌های تابع شامل سه آرگومان x، y و p هستند. آرگومان x به عنوان ورودی داده‌های ورودی الگوریتم یادگیری ماشین (ماتریس ورودی)، آرگومان y به عنوان خروجی‌های مورد انتظار الگوریتم یادگیری ماشین (بردار خروجی) و آرگومان p به عنوان پیش‌بینی‌های الگوریتم یادگیری ماشین (بردار پیش‌بینی) استفاده می‌شود.

در این تابع، ابتدا با استفاده از فرمول محاسبه گرادیان، مقدار ماتریس گرادیان محاسبه می‌شود. برای این کار، از عمل ضرب ترانهاده ماتریس x و تفاضل بین بردار پیش‌بینی و بردار خروجی استفاده می‌شود. سپس، مقدار محاسبه شده بر تعداد داده‌های ورودی (طول بردار y) تقسیم می‌شود تا مقدار نرمال شده گرادیان بدست آید.

در نهایت، مقدار گرادیان به عنوان خروجی از تابع بازگردانده می‌شود و می‌توان از آن در الگوریتم‌های یادگیری ماشین استفاده کرد.

```
def gradient_descent(w , eta , grads):
  w -= eta*grads
  return w
```

تابع gradient_descent در پایتون یک تابع است که از الگوریتم کاهش گرادیان (Gradient Descent) برای به‌روزرسانی وزن‌های یک مدل یادگیری ماشین استفاده می‌کند.

ورودی‌های این تابع شامل وزن‌های فعلی مدل w، نرخ یادگیری eta و گرادیان‌ها grads می‌باشد. ورودی w مشخص می‌کند که وزن‌های مدل در حالت فعلی چه مقداری هستند. نرخ یادگیری نیز نرخی است که مشخص می‌کند چقدر می‌خواهیم وزن‌ها را در هر مرحله به‌روزرسانی کنیم. ورودی grads ماتریس گرادیان است که نشان‌دهنده مقدار گرادیان هر وزن نسبت به تابع هدف است.

در این تابع، ابتدا وزن‌های فعلی مدل با کمک فرمول کاهش گرادیان به‌روزرسانی می‌شوند. برای این کار، از مقدار نرخ یادگیری ضرب شده در گرادیان‌ها کمک گرفته و این مقدار از وزن‌های فعلی کم شده و به‌روزرسانی می‌شوند.

در نهایت، مقدار وزن‌های به‌روزرسانی شده به عنوان خروجی از تابع بازگردانده می‌شود.

```
x_train = np.hstack((np.ones((len(x_train) , 1)) , x_train))
x_train.shape
```

در این کد، ابتدا یک ستون اضافی به آرایه x_train اضافه می‌شود. سپس ابعاد آرایه x_train پس از اضافه کردن این ستون جدید نمایش داده می‌شود.

توضیحات:

- np.ones((len(x_train), 1)): این دستور یک آرایه شامل تعداد len(x_train) سطر و ۱ ستون از مقادیر ۱ ایجاد می‌کند. این کار برای ایجاد ستونی از مقادیر ۱ به منظور اضافه کردن به x_train استفاده می‌شود.
- np.hstack: این تابع برای افزودن آرایه‌ها افقی (افزودن ستون) استفاده می‌شود.
- x_train.shape: این دستور ابعاد آرایه x_train را نمایش می‌دهد.

در نتیجه، ابعاد آرایه x_train پس از اضافه کردن ستون جدید به صورت (تعداد سطر، تعداد ستون) نمایش داده می‌شود:

```
(1096, 5)
```

```
m = 4
w = np.random.randn(m+1 , 1)
print(w.shape)
eta = 0.01
n_epochs = 2000
```

```
(5, 1)
```

```
error_hist = []
for epoch in range(n_epochs):
  p = logistic_regression(x_train , w)
  e = calculate_error(y_train , p)
  error_hist.append(e)
  grads = gradient(x_train , y_train , p)
  w = gradient_descent(w , eta , grads)
  if(epoch + 1) % 100 == 0:
    print(f"Epoch = {epoch} , \t E = {e} \t w={w.T[0]}")
```

این قطعه کد یک حلقه است که مدل رگرسیون لجستیک را آموزش می‌دهد. این حلقه برای تعداد تعیین شده‌ای از epochها اجرا می‌شود و در هر epoch، مقدار خطا، گرادیان و وزن‌های مدل چاپ می‌شود.

توضیحات کامل راجع به کد:

- error_hist: یک لیست که برای ذخیره کردن مقادیر خطا در هر epoch استفاده می‌شود.
- n_epochs: تعداد epochها که در ابتدا تعیین شده است.
- e = calculate_error(y_train, p): با استفاده از تابع calculate_error، مقدار خطا برای داده‌های آموزش محاسبه می‌شود و به لیست error_hist اضافه می‌شود.
- grads = gradient(x_train, y_train, p): با استفاده از تابع gradient، گرادیان مقدار خطا نسبت به وزن‌ها محاسبه می‌شود.

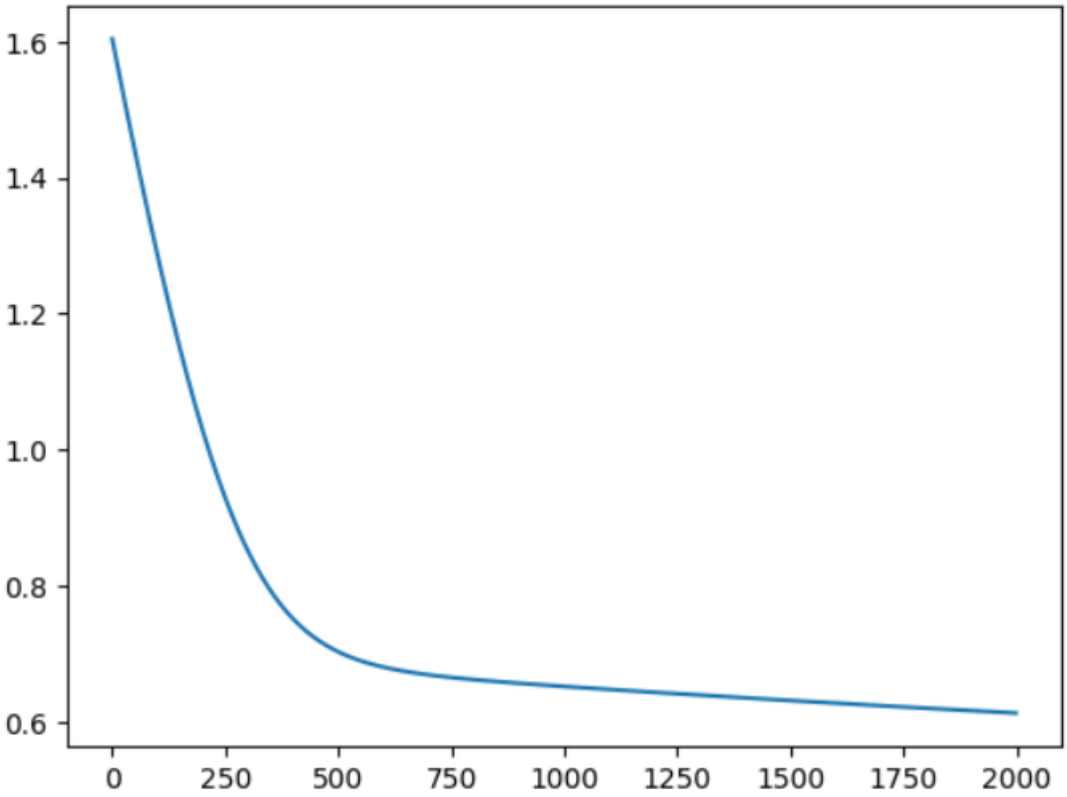
- w = gradient_descent(w, eta, grads): با استفاده از تابع gradient_descent، وزن‌ها با استفاده از گرادیان کاهش می‌یابند.

- if(epoch + 1) % 100 == 0: این شرط برای چاپ مقادیر خطا و وزن‌ها هر ۱۰۰ epoch است.

پس با اجرای این حلقه، مدل رگرسیون لجستیک به صورت تدریجی آموزش داده می‌شود و مقادیر خطا و وزن‌ها در هر ۱۰۰ epoch چاپ می‌شوند:

```
Epoch = 99 ,     E = 0.4803149611507901          w=[ 0.65608943 -1.03567698 -0.04067966  0.26721612  0.01635231]
Epoch = 199 ,    E = 0.1784803033954892          w=[ 0.63564818 -1.04794466 -0.25134719 -0.20747226  0.05812701]
Epoch = 299 ,    E = 0.13314003471670716         w=[ 0.66199394 -1.06949518 -0.34538897 -0.38707848  0.04363592]
Epoch = 399 ,    E = 0.11738489847540806         w=[ 0.69963307 -1.10279446 -0.41426237 -0.47377923  0.01780804]
Epoch = 499 ,    E = 0.10774032154421613         w=[ 0.73951395 -1.13787862 -0.46658178 -0.53106274 -0.0093574 ]
Epoch = 599 ,    E = 0.1006443647252312          w=[ 0.77894075 -1.17158675 -0.50879414 -0.57556507 -0.03417148]
Epoch = 699 ,    E = 0.09503308788578306         w=[ 0.81710365 -1.20318399 -0.54454523 -0.6129377  -0.05576701]
Epoch = 799 ,    E = 0.09041691910043909         w=[ 0.85379937 -1.23265507 -0.57580768 -0.64568427 -0.07424495]
Epoch = 899 ,    E = 0.0865165854065115          w=[ 0.88903345 -1.26018812 -0.60373189 -0.67512105 -0.0899789 ]
Epoch = 999 ,    E = 0.0831546096473055          w=[ 0.92288248 -1.28600967 -0.62904685 -0.70202669 -0.10337891]
Epoch = 1099 ,   E = 0.08021088342884844         w=[ 0.9554429  -1.31033253 -0.65224787 -0.72690425 -0.11481727]
Epoch = 1199 ,   E = 0.07760054515223432         w=[ 0.98681167 -1.33334195 -0.67369054 -0.75010166 -0.12461092]
Epoch = 1299 ,   E = 0.07526162312329585         w=[ 1.01707921 -1.35519486 -0.6936413  -0.77187268 -0.13302308]
Epoch = 1399 ,   E = 0.07314762172858479         w=[ 1.04632738 -1.37602307 -0.71230655 -0.79241029 -0.14027034]
Epoch = 1499 ,   E = 0.07122284845287997         w=[ 1.07462941 -1.39593733 -0.72985038 -0.81186609 -0.1465307 ]
Epoch = 1599 ,   E = 0.06945935605729558         w=[ 1.10205058 -1.4150311  -0.74640604 -0.83036236 -0.15195077]
Epoch = 1699 ,   E = 0.06783487974503954         w=[ 1.12864913 -1.43338372 -0.76208355 -0.84799975 -0.1566518 ]
Epoch = 1799 ,   E = 0.06633140903336593         w=[ 1.15447712 -1.45106301 -0.77697506 -0.86486256 -0.16073455]
Epoch = 1899 ,   E = 0.06493417641540562         w=[ 1.1795813  -1.46812739 -0.79115866 -0.88102235 -0.16428322]
Epoch = 1999 ,   E = 0.06363092663487167         w=[ 1.20400379 -1.48462751 -0.80470117 -0.8965406  -0.16736853]
```

```
plt.plot(error_hist)
```




```
def accuracy(y , y_hat):
  acc = np.sum(y==np.round(y_hat)) / len(y)
  return acc
```

این تابع با نام accuracy، دو آرایه y و y_hat را به عنوان ورودی دریافت می‌کند. سپس دقت مدل را محاسبه می‌کند و مقدار دقت را به عنوان خروجی باز می‌گرداند.

```
x_test = np.hstack((np.ones((len(x_test) , 1)), x_test))
p = logistic_regression(x_test , w)
accuracy(y_test , p)
```

در این کد، ابتدا یک ستون اضافی به آرایه x_test اضافه می‌شود. سپس با استفاده از تابع logistic_regression، پیش‌بینی برای داده‌های تست انجام می‌شود. در نهایت، با استفاده از تابع accuracy، دقت مدل بر روی داده‌های تست محاسبه می‌شود:

```
0.9745454545454545
```

۲_۴:

به طور کلی روش های زیادی برای نرمال سازی داده ها داریم که در این قسمت می خواهیم به دو روش از آن ها اشاره کنیم :

نرمال‌سازی داده‌ها یک مرحله مهم در پردازش و تحلیل داده‌هاست که هدف آن ایجاد یک دسته‌بندی یکنواخت از داده‌ها برای اجتناب از مشکلات ناشی از مقیاس‌ها و واحدهای مختلف در داده‌ها است. دو روش متداول برای نرمال‌سازی داده‌ها عبارتند از:

Min-Max Scaling : در این روش، داده‌ها به گونه‌ای تغییر می‌کنند که حداقل و حداکثر آن‌ها به ترتیب به یک مقدار نگاشته می‌شوند. فرمول نرمال‌سازی Min-Max برای یک داده X به صورت زیر است:

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Z-score Standardization :

در این روش، داده‌ها به گونه‌ای تغییر می‌کنند که میانگین آن‌ها صفر و انحراف معیاری آن‌ها یک شود. فرمول نرمال‌سازی Z-score برای یک داده X به صورت زیر است:

$$X_{\text{standardized}} = \frac{X - \mu}{\sigma}$$

اطلاعات بخش "ارزیابی" در فرآیند نرمال‌سازی به تنهایی معمولاً استفاده نمی‌شود. بخش ارزیابی معمولاً برای ارزیابی عملکرد مدل یا سیستم پس از اعمال تغییرات (مانند نرمال‌سازی) استفاده می‌شود. انتخاب یک روش نرمال‌سازی باید بر اساس نیازها و خصوصیات داده‌ها انجام شود. اگر توزیع داده‌ها نسبت به هم مهم است، ممکن است از Z-score Standardization استفاده کنید. اگر می‌خواهید داده‌ها را به یک بازه خاص نگاشت کنید، Min-Max Scaling مناسب تر است.

در مورد بخش "ارزیابی" در فرآیند عادی سازی، بستگی به زمینه دارد. اگر بخش ارزیابی حاوی اطلاعاتی در مورد دامنه و توزیع ویژگی ها باشد، می تواند در انتخاب روش نرمال سازی مناسب سودمند باشد. به عنوان مثال، اگر ویژگی ها دارای نقاط پرت باشند، عادی سازی امتیاز Z ممکن است قوی تر باشد. اگر ویژگی‌ها محدوده مشخصی دارند، ممکن است مقیاس‌بندی Min-Max ترجیح داده شود. درک ویژگی های داده ها و انتخاب روش عادی سازی بر این اساس ضروری است.

در این قسمت می خواهیم با استفاده از روش اول نرمال سازی را انجام بدهیم. برای این کار باید تمامی داده های داخل یک ستون را که یک ویژگی ما را ایجاد می کنند ، دریافت کرده و از میان این داده ها کمترین داده و بیشترین داده را دریافت کنیم و با استفاده از فرمول min and max scaler مقادیر دیتا ها را نرمال سازی و یا استاندارد سازی کنیم :

```
# part 4
maxx = df[['part1' , 'part2','part3','part4']].max()

minn = df[['part1' , 'part2','part3','part4']].min()

for i in range(4):
  df[f'part{i+1}'] = (df[f'part{i+1}']-minn[i])/(maxx[i]- minn[i])
  print(df[f'part{i+1}'])
```

در این کد، ابتدا برای هر یک از ستون‌های 'part1'، 'part2'، 'part3' و 'part4' از داده‌ها، مقادیر بزرگترین و کوچکترین عناصر را به ترتیب در متغیرهای maxx و minn ذخیره می‌کنیم. سپس با استفاده از یک حلقه، مقادیر هر یک از این ستون‌ها را به مقیاس استاندارد تبدیل می‌کنیم و مقادیر نرمال‌شده را چاپ می‌کنیم:

```
287     0.539656
1030    0.374035
420     0.424926
107     0.731822
923     0.622526
          ...
518     0.757848
908     0.382919
589     0.842099
444     0.737786
1122    0.527044
Name: part1, Length: 1371, dtype: float64
287     0.625706
1030    0.156020
420     0.563842
107     0.872395
923     0.598143
          ...
518     0.772435
908     0.168021
589     0.402788
444     0.387611
1122    0.542780
Name: part2, Length: 1371, dtype: float64
287     0.413923
1030    0.562711
420     0.559463
107     0.044478
923     0.093385
          ...
518     0.182191
908     0.560239
589     0.343050
444     0.347225
1122    0.198470
Name: part3, Length: 1371, dtype: float64
287     0.842331
1030    0.688420
420     0.834217
107     0.413286
923     0.766610
          ...
518     0.710760
908     0.621166
589     0.887476
444     0.758850
1122    0.780484
Name: part4, Length: 1371, dtype: float64
```

```
X = df[['part1' , 'part2','part3','part4']].values

y = df[["part5"]].values
X , y
```

```
(array([[0.53965558, 0.62570581, 0.41392293, 0.84233067],
        [0.37403457, 0.15602046, 0.56271135, 0.68842031],
        [0.42492554, 0.56384169, 0.55946324, 0.83421715],
        ...,
        [0.84209881, 0.40278843, 0.34305038, 0.88747647],
        [0.73778566, 0.38761146, 0.34722468, 0.75884958],
        [0.52704426, 0.54277953, 0.19847029, 0.7804841 ]]),
 array([[0],
        [1],
        [0],
        ...,
        [0],
        [0],
        [1]]))
```

۵_۲:

سایر کدها همانند قسمت قبل هستند:

```
# part 5
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size =
0.2)
x_train.shape , x_test.shape , y_train.shape , y_test.shape
```

```
((1096, 4), (275, 4), (1096, 1), (275, 1))
```

```
x_train = np.hstack((np.ones((len(x_train) , 1)) , x_train))
x_train.shape
```

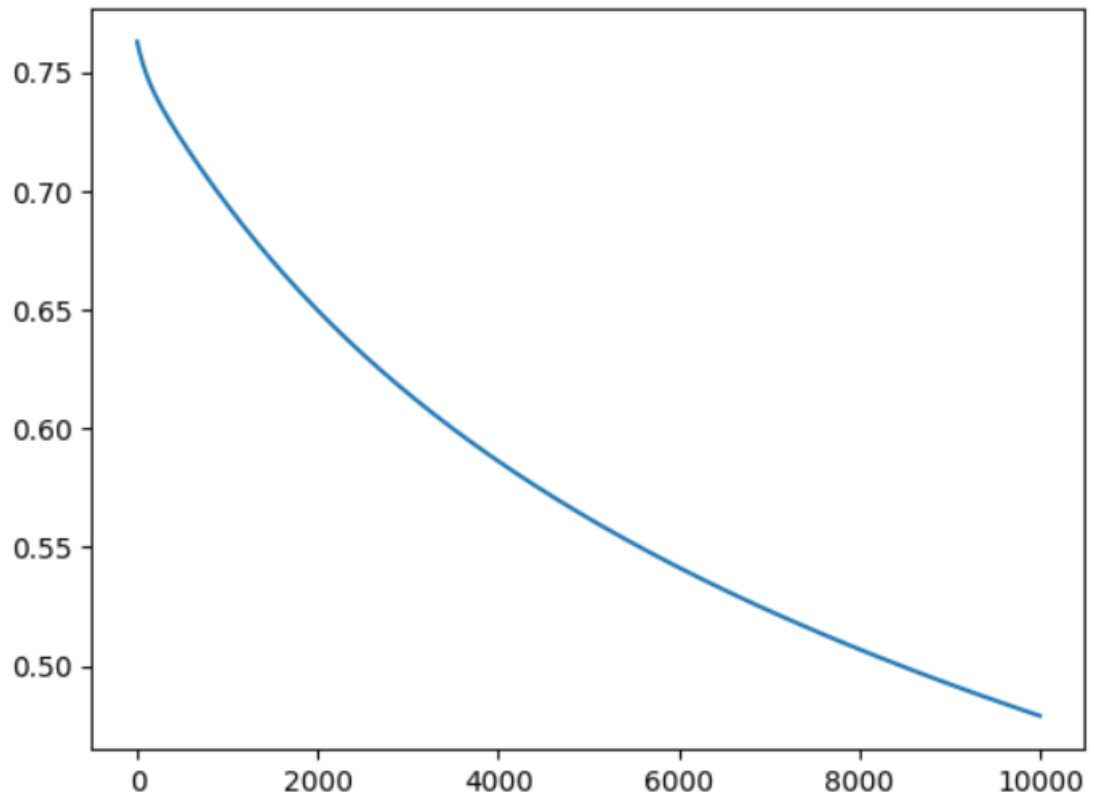
```
(1096, 5)
```

```
m = 4
w = np.random.randn(m+1 , 1)
print(w.shape)
eta = 0.01
n_epochs = 10000
```

```
(5, 1)
```

```
error_hist = []
for epoch in range(n_epochs):
  p = logistic_regression(x_train , w)
  e = calculate_error(y_train , p)
  error_hist.append(e)
  grads = gradient(x_train , y_train , p)
  w = gradient_descent(w , eta , grads)
  if(epoch + 1) % 100 == 0:
    print(f"Epoch = {epoch} , \t E = {e:.4} \t w={w.T[0]}")
```

```
plt.plot(error_hist)
```




```
x_test = np.hstack((np.ones((len(x_test) , 1)), x_test))
x_test.shape
```

```
(275, 5)
```

```
p = logistic_regression(x_test , w)
accuracy(y_test , p)
```

```
0.8654545454545455
```

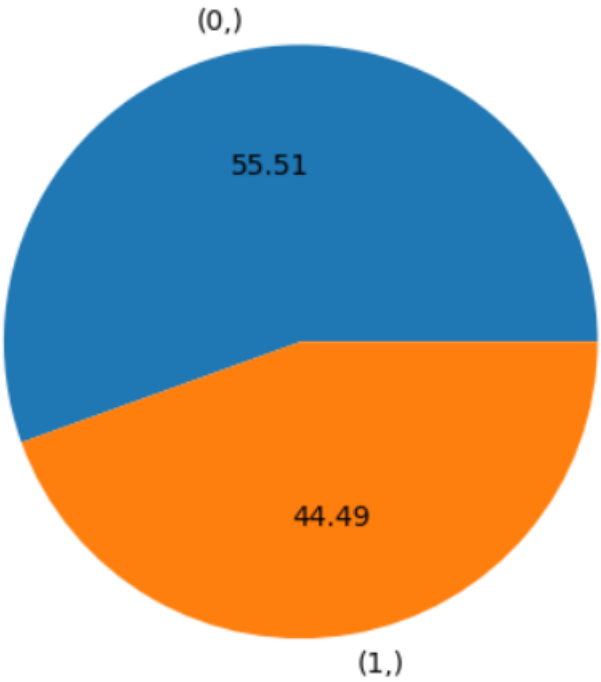
۶_۲:

```
# part 6
```

```
new_y = pd.DataFrame(y, columns=['Column_A'])
new_y.value_counts()
new_y.value_counts().plot.pie(autopct = "%.2f")
```

این کد برای تجسم داده‌ها استفاده می‌شود. ابتدا داده‌های y به یک DataFrame تبدیل می‌شوند و سپس تعداد مقادیر مختلف در آن‌ها محاسبه و نمایش داده می‌شود. در انتها یک نمودار دایره‌ای (Pie chart) از این مقادیر رسم می‌شود. در ضمن دستور new_y.value_counts().plot.pie(autopct = "%.2f") یک نمودار دایره‌ای از مقادیر مختلف را رسم می‌کند و درصد هر بخش را نیز نمایش می‌دهد.

با اجرای این کد، می‌توانیم توزیع مقادیر مختلف در ستون 'Column_A' را به صورت یک نمودار دایره‌ای مشاهده کنیم:



می بینیم که تعادل داده ها وجود ندارد و تعداد نمونه کلاس ها با یکدیگر برابر نیست.

عدم تعادل در دیتاست، به معنای عدم توازن در توزیع کلاس‌ها یا دسته‌های مختلف داده‌ها، می‌تواند به مشکلات مختلفی منجر شود. در زیر چند مشکل اصلی آورده شده است:

۱. مشکل در آموزش مدل:

- در دیتاست‌های ناتوانمند به تعداد نمونه‌های هر کلاس، مدل ممکن است با مشکل مواجه شود. این موضوع می‌تواند منجر به یادگیری ناکافی برای کلاس‌های کم‌نمونه شود.

۲. تاثیرات انحرافی:

- در صورتی که تعداد نمونه‌های یک کلاس زیاد باشد و برای کلاس‌های دیگر کم باشد، مدل ممکن است به سمتی خاص بیفتد و به کلاس‌های کم‌نمونه کمتر توجه کند. این موضوع می‌تواند به انحراف ((biasدر پیش‌بینی‌ها منجر شود.

۳. دقت تخمین‌گر:

- در صورت عدم تعادل، دقت تخمین‌گر برای کلاس‌های با تعداد نمونه بیشتر بالا می‌رود، اما برای کلاس‌های کم‌نمونه کاهش می‌یابد. این ممکن است باعث نادرست فهم شود که مدل بهترین عملکرد را ارائه می‌دهد.

٤. افزایش هزینه آموزش:

- برای مدل‌هایی که با دیتاست‌های ناتوانمند آموزش داده می‌شوند، احتمالاً نیاز به تلاش و هزینه زیادتری برای دستیابی به عملکرد خوب دارند.

٥. اهمال کلاس‌های کم‌نمونه:

- ممکن است مدل به خاطر تعداد کم نمونه‌ها به صورت اشتباهی کلاس‌های کم‌نمونه را اهمال کند یا به اشتباه آنها را به عنوان نمونه‌های کلاس اکثریت (majority class)در نظر بگیرد.

٦. پایداری نتایج:

- دقت و کارایی مدل ممکن است در مواجهه با داده‌های جدید تحت تأثیر قرار گیرد، زیرا مدل ممکن است بر اساس نمونه‌های زیاد یک کلاس و بی‌توجه به کلاس‌های کم‌نمونه باشد.

برای حل مشکلات مربوط به عدم تعادل دیتاست، روش‌هایی مانند oversampling افزایش نمونه‌های کم‌نمونه، undersampling کاهش نمونه‌های بیشتری، یا استفاده از الگوریتم‌های خاص مانند SMOTE معمولاً مورد استفاده قرار می‌گیرند.

حال ما در این قسمت از یکی از این الگوریتم های بالا استفاده می کنیم :

```
! pip install -U imbalanced-learn
```

دستور! "pip install -U imbalanced-learn" در زبان برنامه‌نویسی پایتون برای نصب یا به‌روزرسانی یک کتابخانه خارجی به نام "imbalanced-learn" استفاده می‌شود. این دستور از ابزار pip که یک مدیر بسته‌های استاندارد برای پایتون استفاده می‌کند.

```
from imblearn.under_sampling import RandomUnderSampler

y = pd.DataFrame(y, columns=[''])
rus = RandomUnderSampler(sampling_strategy=1)
x_res_undersampling , y_res_undersampling = rus.fit_resample(X , y)
ax = y_res_undersampling.value_counts().plot.pie(autopct = '%.2f')
_ = ax.set_title("under sampling")
```

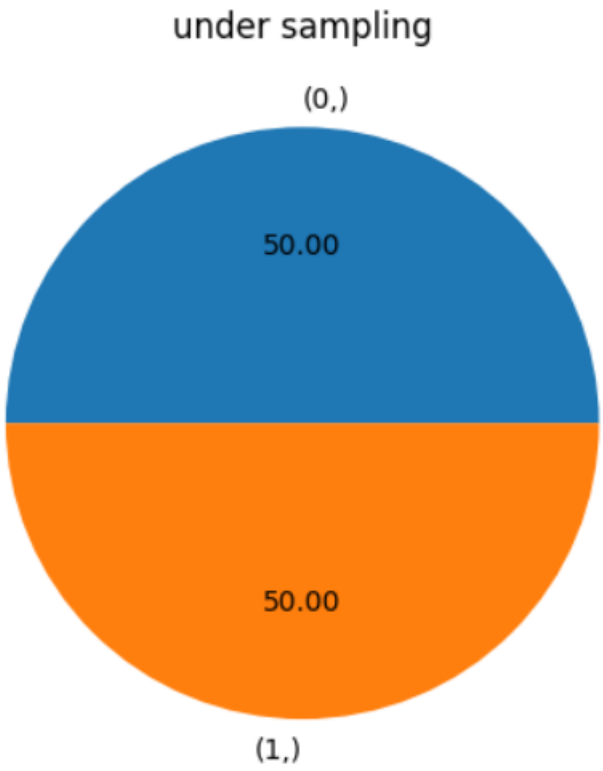
در این بخش از کد، از کتابخانه imbalanced-learn برای اعمال روش زیرنمونه‌برداری تصادفی (Random Under Sampling) استفاده شده است. این روش به منظور کاهش تعداد نمونه‌های کلاس اکثریت و ایجاد تعادل بین کلاس‌ها در مسائل داده‌کاوی و یادگیری ماشین استفاده می‌شود.

در این کد، ابتدا متغیر y به صورت یک دیتافریم تبدیل شده است. سپس یک شیء از کلاس RandomUnderSampler با استراتژی نمونه‌برداری ۱ ایجاد شده است. در اینجا، استراتژی نمونه‌برداری ۱ به این معناست که تعداد نمونه‌های کلاس اقلیت برابر با تعداد نمونه‌های کلاس اکثریت قرار می‌گیرد.

سپس این شیء بر روی داده‌های X و y اعمال شده است. این عمل باعث می‌شود تعداد نمونه‌های کلاس اکثریت کاهش یابد و تعداد نمونه‌های کلاس اقلیت برابر با تعداد نمونه‌های کلاس اکثریت شود.

در ادامه، تعداد مقادیر مختلف در y_res_undersampling شمارش شده و یک نمودار دایره‌ای از توزیع مقادیر ایجاد شده است. این نمودار به صورت خودکار نسبت تعداد نمونه‌های هر کلاس را نشان می‌دهد. به عنوان مثال، اگر کلاس اکثریت ۷۰۰۰ نمونه داشته باشد و کلاس اقلیت ۳۰۰۰ نمونه داشته باشد، پس این نمودار ۲ دسته را با نسبت ۷:۳ نشان می‌دهد.

در نهایت، باید توجه کرد که اگر تعداد نمونه‌های کلاس‌ها کم باشد، ممکن است نمودار دایره‌ای به درستی نمایش داده نشود. بنابراین اگر تعداد نمونه‌ها کمتر از ۱۰۰۰ نمونه باشد، بهتر است از روش‌های دیگر برای نمایش توزیع استفاده کنیم.



می‌بینیم که داده‌ها به خوبی متعادل شده‌اند.

۷_۲:

```
# part 7

from sklearn.linear model import LogisticRegression
```

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X = x_res_undersampling
y = y_res_undersampling
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size =
0.2)
```

```
model = LogisticRegression()
model.fit(X , y)
```

```
y_hat = model.predict(x_test)
model.score(x_test , y_test)
y_test.shape ,
y_hat = y_hat.reshape(244 , 1)
y_test.shape , y_hat.shape

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_hat)
score
```

ابتدا، با استفاده از مدل آموزش داده شده، پیش‌بینی‌های مربوط به داده‌های آزمون انجام شد. سپس دقت پیش‌بینی مدل بر روی داده‌های آزمون محاسبه شد که به مقدار score منجر شد. ابعاد آرایه‌های y_test و y_hat بررسی شده و سپس آرایه y_hat به شکل جدیدی با ابعاد ۲۴۴ در ۱ تغییر شکل داده شد. در نهایت، با استفاده از متد accuracy_score از کتابخانه sklearn.metrics، دقت پیش‌بینی مدل بر روی داده‌های آزمون محاسبه و در متغیر score ذخیره شد.

مشاهده خروجی:

```
0.9795081967213115
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression , SGDClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

X = df[['part1' , 'part2','part3','part4']].values
```

```
y = df[["part5"]].values
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size =
0.2)
model = LogisticRegression(random_state = 93, solver='sag', max_iter=200)
model.fit(X , y)
y_hat = model.predict(x_test)
model.score(x_test , y_test)
```

مشاهده خروجی:

```
0.9818181818181818
```

سوال ۳:

۳_۱:

با توجه به اطلاعاتی که در خود سایت این دیتاست نوشته شده است می توان فهمید که این مجموعه داده شامل شاخص های مختلف مرتبط با سلامت برای نمونه ای از افراد است. در اینجا توضیح مختصری از هر ستون آورده شده است:

Heart Diseaseor Attack : نشان می دهد که آیا فرد دچار بیماری قلبی یا حمله قلبی شده است (دودویی: ۰ = خیر، ۱ = بله).

HighBP : وضعیت فشار خون بالا (باینری: ۰ = خیر، ۱ = بله).

HighChol : وضعیت کلسترول بالا (دودویی: ۰ = خیر، ۱ = بله).

CholCheck : دفعات بررسی کلسترول (طبقه ای).

BMI : شاخص توده بدن (مستمر).

Smoker: وضعیت سیگار کشیدن (دودویی: ۰ = خیر، ۱ = بله).

Stroke: سابقه سکته مغزی (باینری: ۰ = خیر، ۱ = بله).

Diabetes: وضعیت دیابت (دودویی: ۰ = خیر، ۱ = بله).

PhysActivity : سطح فعالیت بدنی (طبقه ای).

Fruits: فراوانی مصرف میوه (قسمتی).

Veggies: فراوانی مصرف سبزیجات (قسمتی).

HvyAlcoholConsump : وضعیت مصرف الکل سنگین (باینری: ۰ = خیر، ۱ = بله).

AnyHealthcare : دسترسی به هر مراقبت بهداشتی (باینری: ۰ = خیر، ۱ = بله).

NoDocbcCost : بدون پزشک به دلیل هزینه (باینری: ۰ = خیر، ۱ = بله).

GenHlth : ارزیابی سلامت عمومی (طبقه ای).

MentHlth : ارزیابی سلامت روان (مقوله ای).

PhysHlth : ارزیابی سلامت جسمانی (طبقه ای).

DiffWalk : وضعیت دشواری راه رفتن (باینری: ۰ = خیر، ۱ = بله).

Sex: جنسیت فرد (دودویی: ۰ = زن، ۱ = مرد).

Age: سن فرد (مستمر).

Education: مقطع تحصیلی (قسمتی).

Income: سطح درآمد (مقوله ای).

این مجموعه داده حاوی انواع اطلاعات مرتبط با سلامتی، عوامل سبک زندگی و اطلاعات جمعیتی برای گروهی از افراد است که آن را برای بررسی همبستگی ها و عوامل خطر بالقوه بیماری قلبی و سایر شرایط سلامتی مناسب می کند.

```
# part 1
!pip install --upgrade --no-cache-dir gdown
!gdown 1_aCSZ-4ROIAFaUME8bO3vgYx9iCajaI1
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression , SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

```
df=pd.read_csv('/content/heart_disease_health_indicators.csv')

df1 = df[df["HeartDiseaseorAttack"] == 1]
df1 = df1.iloc[0:100, :]

df2 = df[df["HeartDiseaseorAttack"] == 0]
df2 = df2.iloc[0:100, :]

df = pd.concat([df1, df2], ignore_index=True)
```

```
df = shuffle(df)

df = df.reset_index(drop=True)

df
```

در این بخش از کد، ابتدا داده‌ها از یک فایل CSV وارد شده‌اند. سپس دو زیرمجموعه از داده‌ها بر اساس مقدار ستون " "HeartDiseaseorAttackانتخاب شده‌اند. سپس از هرکدام از این دو زیرمجموعه، ۱۰۰ نمونه اول انتخاب شده و سپس با استفاده از تابع concat این دو زیرمجموعه به یکدیگر اضافه شده‌اند. سپس داده‌ها تصادفی مخلوط شده و ایندکس‌ها مجدداً تنظیم شده‌اند. در نهایت، داده‌های ترکیب‌شده و بازنشانی شده در df ذخیره شده‌اند.

مشاهده خروجی اکسل در پایتون:

index	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age
0	1	1	1	1	23	1	0	0	1	1	1	0	1	0	2	0	0	0	0	9
1	1	0	0	1	27	1	0	0	1	0	1	0	1	0	3	0	30	0	1	9
2	0	1	0	1	27	1	0	0	0	0	0	0	1	0	4	0	5	1	0	11
3	0	1	1	1	23	0	1	0	1	1	1	0	1	0	3	0	0	1	1	13
4	0	0	0	1	31	1	0	0	1	0	0	0	1	0	2	0	4	0	1	6
5	1	1	1	1	26	1	0	2	0	1	0	0	1	0	4	0	0	0	0	10
6	0	0	1	1	22	0	0	0	1	1	1	0	1	0	3	30	0	0	0	8
7	1	0	1	1	21	1	0	0	1	1	1	0	1	0	3	0	0	0	0	13
8	0	1	1	1	33	1	0	2	0	0	1	0	1	0	3	0	30	1	0	11
9	1	1	1	1	17	1	0	0	1	0	1	0	1	0	4	30	0	1	0	8
10	1	1	1	1	28	0	0	0	0	0	1	0	1	0	4	30	30	1	0	6
11	1	1	1	1	37	0	1	2	1	1	0	0	1	1	3	0	0	1	0	8
12	1	0	1	1	26	1	0	0	1	1	1	0	1	0	2	0	0	0	0	11
13	1	0	1	1	25	1	0	2	1	1	1	0	1	0	2	0	7	0	1	10
14	1	1	1	1	28	1	0	2	0	0	1	1	1	0	3	0	30	0	1	10
15	0	1	1	1	32	1	0	0	0	0	1	0	1	0	4	5	15	0	0	10
16	1	1	1	1	30	1	0	0	1	0	0	0	1	0	4	10	17	1	0	9
17	0	1	0	1	31	0	0	2	0	1	0	0	1	0	3	0	5	0	0	13
18	1	1	0	1	27	0	1	2	0	1	1	0	1	0	5	30	30	1	0	12
19	1	1	1	1	32	1	0	2	1	1	1	0	1	0	3	30	10	1	0	11
20	1	1	1	1	33	1	0	0	1	0	1	0	1	0	3	1	0	0	1	11
21	1	1	0	1	29	1	1	2	1	0	0	0	1	0	4	0	30	1	0	10
22	0	0	0	1	28	1	0	0	1	1	1	0	1	1	3	30	30	1	1	3
23	1	1	1	1	28	1	0	2	0	0	1	0	1	0	4	0	0	0	1	12
24	1	1	1	1	22	0	1	0	0	1	0	0	1	0	3	30	0	1	0	12

۲_۳:

```
# part 2
X = df[df.columns[1:]].values

y = df[["HeartDiseaseorAttack"]].values
X ,y
```

در این بخش از کد، داده‌های ورودی X و متغیر پاسخ y ایجاد شده‌اند. برای ایجاد متغیر ورودی X ، تمام ستون‌های داده‌های فراخوانی شده به جز ستون اول (که به عنوان متغیر پاسخ استفاده شده است) به عنوان متغیر ورودی انتخاب شده‌اند. متغیر پاسخ y نیز مقدار ستون "HeartDiseaseorAttack" انتخاب شده است. در نهایت، مقادیر متغیرهای ورودی X و متغیر پاسخ y چاپ شده‌اند:

```
(array([[ 1,  1,  1, ...,  9,  6,  7],
        [ 0,  0,  1, ...,  9,  4,  8],
        [ 1,  0,  1, ..., 11,  4,  2],
        ...,
        [ 0,  0,  0, ...,  7,  6,  1],
        [ 0,  0,  1, ..., 11,  6,  7],
        [ 1,  1,  1, ..., 10,  4,  3]]),
 array([[1],
        [1],
        [0],
        [0],
        [0],
        [1],
        [0],
        [1],
        [0],
        [1],
        [1],
```

۳_۳:

طبق دستوراتی که می‌دانیم ادامه می‌دهیم:

```
# part 3
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
x_train.shape , x_test.shape , y_train.shape , y_test.shape
```

```
((160, 21), (40, 21), (160, 1), (40, 1))
```

```
logreg = LogisticRegression()
```

```
logreg.fit(x_train, y_train)

y_pred = logreg.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print("y_pred:", y_pred.shape)
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion_mat)
print("Classification Report:")
print(classification_rep)
```

در این بخش از کد، یک مدل رگرسیون لجستیک (Logistic Regression) ایجاد شده و با داده‌های آموزش x_train و y_train آموزش داده شده است. سپس پیش‌بینی‌های مدل بر روی داده‌های آزمون x_test انجام شده و سپس معیارهای عملکرد مدل از جمله دقت، ماتریس اشتباهات (confusion matrix) و گزارش طبقه‌بندی (classification report) محاسبه و چاپ شده‌اند.

مشاهده خروجی:

```
y_pred: (40,)
Accuracy: 0.75
Confusion Matrix:
[[13  3]
 [ 7 17]]
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.81      0.72        16
           1       0.85      0.71      0.77        24

    accuracy                           0.75        40
   macro avg       0.75      0.76      0.75        40
weighted avg       0.77      0.75      0.75        40
```

```
logreg.score(x_test, y_test)
```

در این بخش از کد، از متد score برای محاسبه دقت مدل رگرسیون لجستیک بر روی داده‌های آزمون استفاده شده است. این متد به صورت خودکار پیش‌بینی‌ها را انجام می‌دهد و سپس دقت مدل را بر روی داده‌های آزمون محاسبه می‌کند.

```
0.75
```

و همچنین به طور مشابه:

```
logreg.score(x_train , y_train)
```

```
0.70625
```

```
print("Training Accuracy:", logreg.score(x_train, y_train))
print("Test Accuracy:", logreg.score(x_test, y_test))
```

```
Training Accuracy: 0.70625
Test Accuracy: 0.75
```

۴_۳:

```
from sklearn.metrics import log_loss

#loss = log_loss(y_pred, y_test)
loss = log_loss(y_test,y_pred)
print("Loss:", loss)
```

در این بخش از کد، از متد log_loss برای محاسبه مقدار تابع هزینه لجستیک بر روی داده‌های آزمون استفاده شده است. برای محاسبه تابع هزینه، پیش‌بینی‌های مدل بر روی داده‌های آزمون y_pred و برچسب‌های واقعی این داده‌ها y_test به عنوان ورودی به متد داده شده‌اند.

مشاهده خروجی:

```
Loss: 9.010913347279288
```

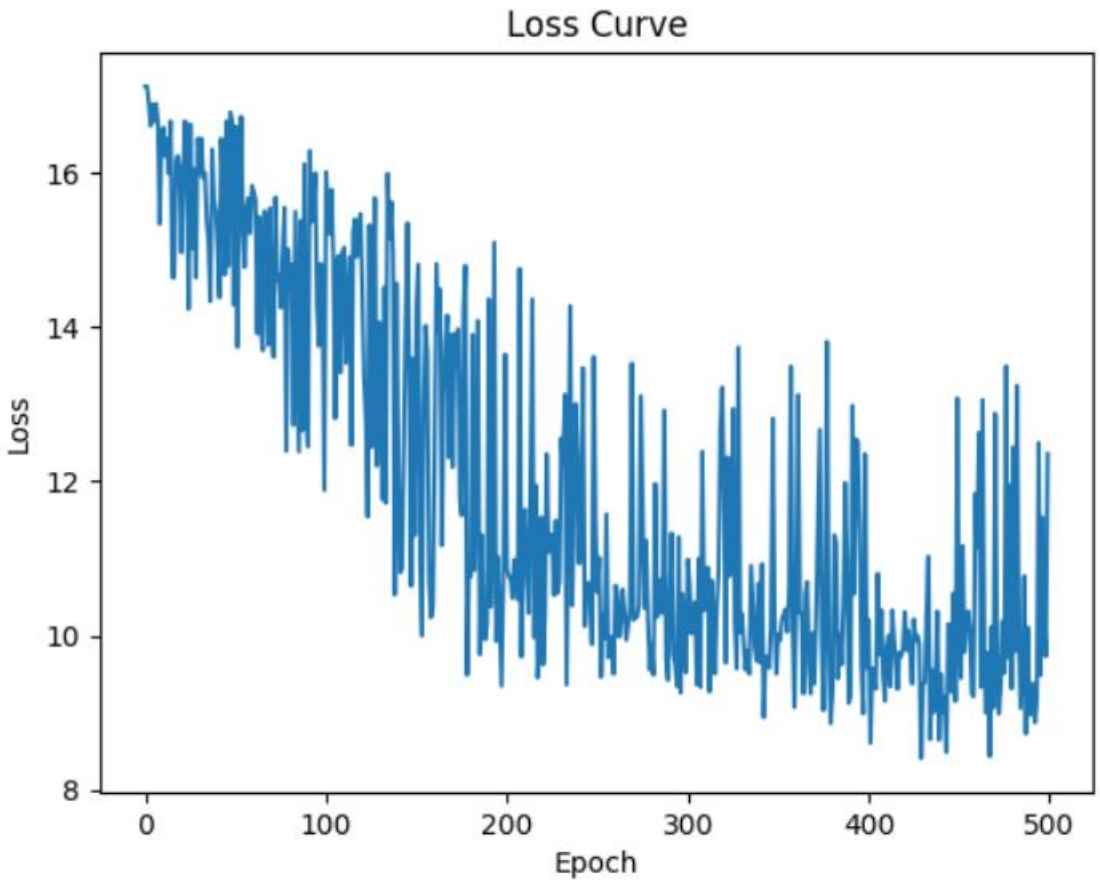
بررسی نمودار تابع خطا:

```
model = SGDClassifier(loss='log', random_state=93)
losses = []
epochs = 500
```

```
for _ in range(epochs):

    model.partial_fit(x_train, y_train, [0, 1])
    loss = log_loss(y_train , model.predict_proba(x_train))
    losses.append(loss)

plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.show()
```

در این بخش از کد، یک مدل SGDClassifier با استفاده از تابع هزینه لجستیک loss='log' ایجاد شده است. سپس مدل با استفاده از متد partial_fit بر روی داده‌های آموزش x_train و y_train آموزش داده می‌شود. همچنین برای هر epoch، مقدار تابع هزینه بر روی داده‌های آموزش محاسبه شده و در لیست losses ذخیره می‌شود. در نهایت، نمودار مقادیر تابع هزینه بر حسب epoch رسم می‌شود.

مشاهده نمودار تابع هزینه:



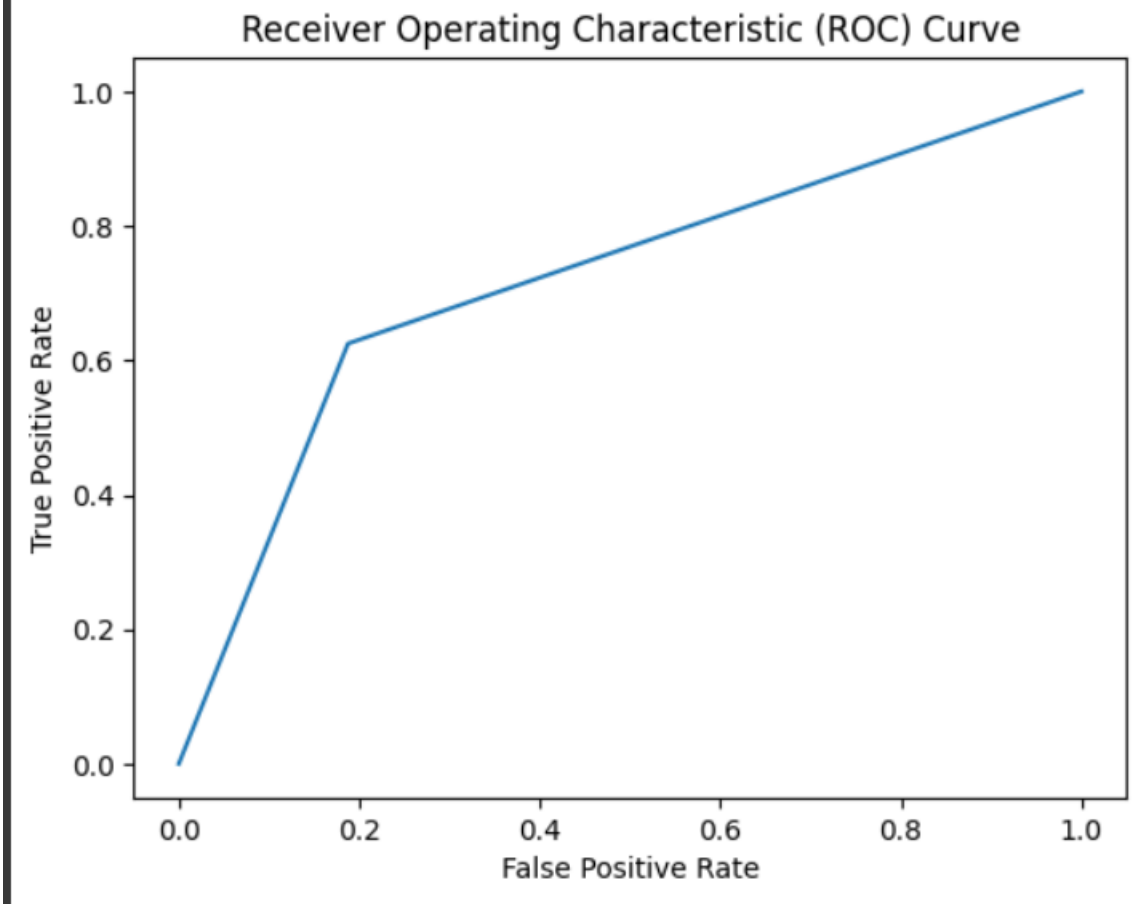
۳_۵:

```
# part 5
from sklearn.metrics import roc_auc_score, roc_curve

auc = roc_auc_score(y_pred, y_test)
print("AUC:", auc)

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.show()
```

این بخش از کد برای محاسبه و نمایش منحنی ROC و محاسبه مساحت زیر آن (AUC) برای ارزیابی عملکرد مدل استفاده می‌شود. منحنی ROC یک ابزار ارزیابی برای مدل‌های دسته‌بندی است که نشان می‌دهد چقدر مدل ما توانایی تفکیک بین دو کلاس مختلف را دارد. در اینجا، ابتدا با استفاده از متد roc_auc_score، مقدار AUC برای پیش‌بینی‌های مدل و برچسب‌های واقعی داده‌های آزمون محاسبه می‌شود. سپس با استفاده از متد roc_curve، نقاط منحنی ROC برای پیش‌بینی‌های مدل و برچسب‌های واقعی محاسبه می‌شود. سپس این نقاط در یک نمودار رسم شده و منحنی ROC نمایش داده می‌شود. این نمودار نشان می‌دهد که مدل چقدر توانایی تفکیک بین دو کلاس مختلف را دارد.

نمایش خروجی:

```
AUC: 0.71875
```



پایان